Metode Numerik Menggunakan R

Bakti Siregar, M.Sc, Trisha M.A. Januaviani, M.Mat & Dr. Kalfin, M.Mat

06 Maret 2023

Contents

Kata Pengantar

Modul ini dirancang sebagai panduan bagi mahasiswa untuk melakukan praktikum Metode Numerik di Universitas Matana. Beberapa topik penting dalam analisis numerik dibahas dalam buku ini, seperti interpolasi, integrasi numerik, solusi persamaan nonlinear, dan pemecahan sistem persamaan linear. Modul ini dirancang agar mahasiswa dapat memahami konsep dan menerapkan dengan mudah dan praktis melalui studi kasus yang relevan dengan alat bantu bahasa pemrograman R. Setelah mempelajari modul ini, Mahasiswa (Pembaca) diharapkan untuk dapat mengembangkan keterampilan komputasi numerik mereka dalam proses analisis dan penyelesaian masalah, khususnya dalam perhitungan Matematika/Statistika yang lebih kompleks.

Ringkasan

Adapun isi pembahasan dalam modul ini adalah sebagai berikut:

- Pengantar metode numerik
- Aljabar linier
- Akar persamaan non-linier
- Interpolasi dan ekstrapolasi
- Diferensiasi dan integrasi numerik
- Persamaan diferensial
- Analisis data

Catatan: Sebelum mempelajari dan menggunakan modul ini Mahasiswa (Pengguna) diharapkan sudah mempunyai pengalaman yang cukup tentang dasar pemrograman R dan mampu untuk melakukan visualisasi data dengan bahasa pemrograman R.

Tim Penyusun

Buku ini tidaklah sempurna. Masih terdapat banyak kekurangan dalam buku ini. Namun, kekurangan tersebut tetaplah dapat diperbaiki. Penulis membuka ruang seluas-luasnya dalam melakukan perbaikan dalam buku ini bagi pembaca. Untuk dapat melakukannya pembaca hanya perlu melakukan *pull request* ke akun github penulis atau pembaca dapat melakukannya melalui email yang selanjutnya akan penulis jadikan acuan perbaikan.

Github : https://github.com/dsciencelabs/Metode_Numerik_R Email : dsciencelabs@outlook.com

Ucapan Terima Kasih

Buku ini tidak akan ada apabila tidak adanya inspirasi dari rekan-rekan Teknik Lingkungan ITB dan paket `bookdown` dari Rstudio. Ucapan terima kasih juga penulis sampaikan komunitas `R` Idonesia yang telah memberikan banyak pelajaran bagi penulis dalam belajar `R`. Tidak pula penulis mengucapkan terima kasih pada pengembang paket-paket metode numerik yang karyanya telah membantu saya dalam menyusun buku ini.

Lisensi

Chapter 1

Bahasa Pemrograman R

Dewasa ini tersedia banyak sekali *software* yang dapat digunakan untuk membantu kita dalam melakukan analisa data. *software* yang digunakan dapat berupa *software* berbayar atau gratis.

`R` merupakan merupakan salah satu *software* gratis yang sangat populer di Indonesia. Kemudahan penggunaan serta banyaknya besarnya dukungan komunitas membuat `R` menjadi salah satu bahasa pemrograman paling populer di dunia.

Paket yang disediakan untuk analisis statistika dan analisa numerik juga sangat lengkap dan terus bertambah setiap saat. Hal ini membuat `R` banyak digunakan oleh para analis data.

Pada *chapter* ini penulis akan memperkenalkan kepada pembaca mengenai bahasa pemrograman `R`. Mulai dari sejarah, cara instalasi sampai dengan bagaimana kita memanfaatkan fitur dasar bantuan untuk menggali lebih jauh tentang fungsi-fungsi `R`

1.1 Sejarah R

`R` Merupakan bahasa yang digunakan dalam komputasi **statistik** yang pertama kali dikembangkan oleh **Ross Ihaka** dan **Robert Gentlement** di University of Auckland New Zealand yang merupakan akronim dari nama depan kedua pembuatnya. Sebelum `R` dikenal ada `S` yang dikembangkan oleh **John Chambers** dan rekan-rekan dari **Bell Laboratories** yang memiliki fungsi yang sama untuk komputasi statistik. Hal yang membedakan antara keduanya adalah `R` merupakan sistem komputasi yang bersifat gratis.Logo `R` dapat dilihat pada Gambar 1.1.

Figure 1.1: Logo R.

`R` dapat dibilang merupakan aplikasi sistem **statistik** yang kaya. Hal ini disebabkan banyak sekali Paket yang dikembangkan oleh pengembang dan komunitas untuk keperluan analisa statistik seperti *linear regression*, *clustering*, *statistical test*, dll. Selain itu, `R` juga dapat ditambahkan Paket-Paket lain yang dapat meningkatkan fiturnya.

Sebagai sebuah bahasa pemrograman yang banyak digunakan untuk keperluan analisa data, `R` dapat dioperasikan pada berbagai sistem operasi pada komputer. Adapun sistem operasi yang didukung antara lain: `UNIX`, `Linux`, `Windows`, dan `MacOS`.

1.2 Fitur dan Karakteristik R

`R` memiliki karakteristik yang berbeda dengan bahasa pemrograman lain seperti `C++`,`python`, dll. `R` memiliki aturan/sintaks yang berbeda dengan bahasa pemrograman yang lain yang membuatnya memiliki ciri khas tersendiri dibanding bahasa pemrograman yang lain.

Beberapa ciri dan fitur pada `R` antara lain:

1. **Bahasa `R` bersifat case sensitif**. maksudnya adalah dalam proses input `R` huruf besar dan kecil sangat diperhatikan. Sebagai contoh kita ingin melihat apakah objek A dan B pada sintaks berikut:

```
A <- "Andi"
B <- "andi"

# cek kedua objek A dan B
A == B
```

```
## [1] FALSE
```

```
# Kesimpulan : Kedua objek berbeda
```

2. **Segala sesuatu yang ada pada program `R` akan diangap sebagai objek**. konsep objek ini sama dengan bahasa pemrograma berbasis objek yang lain seperti `Java`, `C++`, `python`, dll.Perbedaannya adalah bahasa `R` relatif lebih sederhana dibandingkan bahasa pemrograman berbasis obejk yang lain.
3. **interpreted language atau script**. Bahasa `R` memungkinkan pengguna untuk melakukan kerja pada `R` tanpa perlu kompilasi kode program menjadi bahasa mesin.
4. Mendukung proses **loop**, **decision making**, dan menyediakan berbagai jenis **operstor** (aritmatika, logika, dll).
5. **Mendukung export dan import berbagai format file**, seperti:TXT, CSV, XLS, dll.
6. **Mudah ditingkatkan melalui penambahan fungsi atau *library***. Penambahan Paket dapat dilakukan secara online melalui CRAN atau melalui sumber seperti github.
7. **Menyedikan berbagai fungsi untuk keperluan visualisasi data**. Visualisasi data pada `R` dapat menggunakan Paket bawaan atau Paket lain seperti `ggplo2`,`ggvis`, dll.

1.3 Kelebihan dan Kekurangan R

Selain karena `R` dapat digunakan secara gratis terdapat **kelebihan** lain yang ditawarkan, antara lain:

1. **Protability**. Penggunaan software dapat digunakan kapanpun tanpa terikat oleh masa berakhirnya lisensi.
2. **Multiplatform**. `R` bersifat *Multiplatform Operating Systems*, dimana *software* `R` lebih kompatibel dibanding *software* statistika lainnya. Hal in berdampak pada kemudahan dalam penyesuaian jika pengguna harus berpindah sistem operasi karena `R` baik pada sistem operasi seperti `windows` akan sama pengoperasiannya dengan yang ada di `Linux` (Paket yang digunakan sama).
3. **General** dan **Cutting-edge**. Berbagai metode statistik baik metode klasik maupun baru telah diprogram kedalam `R`. Dengan demikian *software* ini dapat digunakan untuk analisis statistika dengan pendekatan klasik dan pendekatan modern.
4. **Programable**. Pengguna dapat memprogram metode baru atau mengembangakan modifikasi dari analisis statistika yang telah ada pada sistem `R`.
5. **Berbasis analisis matriks**. Bahasa `R` sangat baik digunakan untuk *programming* dengan basis matriks.

6. Fasiltas grafik yang lengkap.

Adapun kekurangan dari `R` antara lain:

1. **Point and Click GUI**. Interaksi utama dengan `R` bersifat *CLI* (*Command Line Interface*), walaupun saat ini telah dikembangkan Paket yang memungkinkan kita berinteraksi dengan `R` menggunakan *GUI* (*Graphical User Interface*) sederhana menggunakan Paket `R-Commander` yang memiliki fungsi yang terbatas. `R- Commander` sendiri merupakan *GUI* yang diciptakan dengan tujuan untuk keperluan pengajaran sehingga analisis statistik yang disediakan adalah yang klasik. Meskipun terbatas Paket ini berguna jika kita membutuhkan analisis statistik sederhana dengan cara yang simpel.
2. **Missing statistical function**. Meskipun analisis statistika dalam `R` sudah cukup lengkap, namun tidak semua metode statistika telah diimplementasikan ke dalam `R`. Namun karena `R` merupakan *lingua franca* untuk keperluan komputasi statistika modern staan ini, dapat dikatakan ketersediaan fungsi tambahan dalam bentuk Paket hanya masalah waktu saja.

1.4 RStudio

Aplikasi `R` pada dasarnya berbasis teks atau *command line* sehingga pengguna harus mengetikkan perintah-perintah tertentu dan harus hapal perintah-perintahnya. Setidaknya jika kita ingin melakukan kegiatan analisa data menggunakan `R` kita harus selalu siap dengan perintah-perintah yang hendak digunakan sehingga buku manual menjadi sesuatu yang wajib adasaat berkeja dengan `R`.

Kondisi ini sering kali membingunkan bagi pengguna pemula maupun pengguna mahir yang sudah terbiasa dengan aplikasi statistik lain seperti SAS, SPSS, Minitab, dll. Alasan itulah yang menyebabkan pengembang `R` membuat berbagai *frontend* untuk `R` yang berguna untuk memudahkan dalam pengoperasian `R`.

`RStudio` merupakan salah satu bentuk *frontend* `R` yang cukup populer dan nyaman digunakan. Selain nyaman digunakan, `RStudio` memungkinkan kita melakukan penulisan laporan menggunakan `Rmarkdown` atau `RNotebook` serta membuat berbagai bentuk project seperti shyni, dll. Pada `R` studio juga memungkinkan kita mengatur *working directory* tanpa perlu mengetikkan sintaks pada Commander, yang diperlukan hanya memilihnya di menu `RStudio`. Selain itu, kita juga dapat meng-import file berisikan data tanpa perlu mengetikkan pada Commander dengan cara memilih pada menu `Environment`.

1.5 Menginstall R dan RStudio

Pada tutorial ini hanya akan dijelaskan bagaimana menginstal `R` dan `RStudio` pada sistem operasi `windows`. Sebelum memulai menginstal sebaiknya pembaca mengunduh terlebih dahulu *installer* R dan RStudio.

1. Jalankan proses pemasangan dengan meng-klik *installer* aplikasi `R` dan `RStudio`.
2. Ikuti langkah proses pemasangan aplikasi yang ditampilkan dengan klik `OK` atau `Next`.
3. Apabila pemasangan telah dilakukan, jalankan aplikasi yang telah terpasang untuk menguji jika aplikasi telah berjalan dengan baik.

Jendela aplikasi yang telah terpasang ditampilkan pada Gambar 1.2 dan Gambar 1.3.

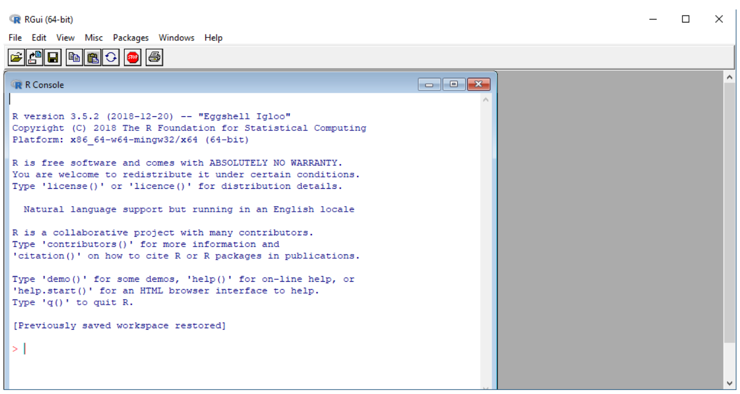

Figure 1.2: Jendela R.

Tips: Sebaiknya install `R` terlebih dahulu sebelum `RStudio`

1.6 Working Directory

Setiap pengguna akan bekerja pada tempat khusus yang disebut sebagai *working directory*. *working directory* merupakan sebuah folder dimana `R` akan membaca dan menyimpan file kerja kita. Pada pengguna `windows`, *working directory* secara default pada saat pertama kali menginstall `R` terletak pada folder `c:\\Document`.

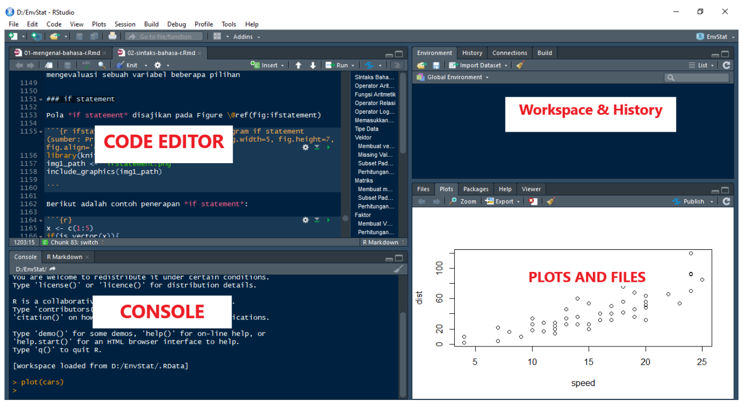

Figure 1.3: Jendela RStudio.

1.6.1 Mengubah Lokasi Working Directory

Kita dapat mengubah lokasi *working directory* berdasarkan lokasi yang kita inginkan, misalnya letak data yang akan kita olah tidak ada pada folder default atau kita ingin pekerjaan kita terkait `R` dapat berlangsung pada satu folder khusus.

Berikut adalah cara mengubah *working directory* pada `R`.

1. Buatlah folder pada drive (kita bisa membuat folder pada selain drive c) dan namai dengan nama yang kalian inginkan. Pada tutorial ini penulis menggunakan nama folder `R`.
2. Jika pengguna menggunakan `RStudio`, pada menu `RStudio` pilih **Session > Set Working Directory > Chooses Directory**. Proses tersebut ditampilkan pada Gambar 1.4
3. Pilih folder yang telah dibuat pada step 1 sebagai *working directory.

> **Penting:** Data atau file yang hendak dibaca selama proses kerja pada `R` harus selalu diletakkan pada working directory. Jika tidak maka data atau file tidak akan terbaca.

Untuk mengecek apakah proses perubahan telah terjadi, kita dapat mengeceknya dengan menjalankan perintah berikut untuk melihat lokasi *working directory* kita yang baru.

```
getwd()
```

Selain itu kita dapat mengubah *working directory* menggunakan perintah berikut:

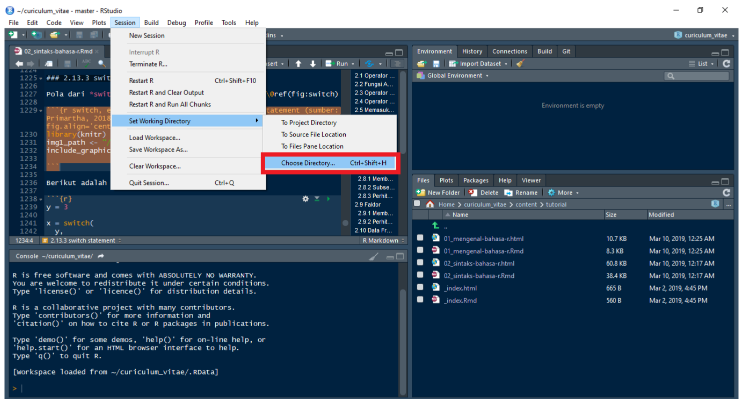

Figure 1.4: Mengubah working directory.

```r
# Ubah working directori pada folder R
setwd("/Documents/R")
```

Peringatan !!!

Pada proses pengisian lokasi folder pastikan pemisah pada lokasi folder menggunakan tanda “/” bukan “”

1.6.2 Mengubah Lokasi Working Directory Default

Pada proses yang telah penulis jelaskan sebelumnya. Proses perubahan *working directory* hanya berlaku pada saat pekerjaan tersebut dilakukan. Setelah pekerjaan selesai dan kita menjalankan kembali `R` maka *working directory* akan kembali secara default pada working directory lama.

Untuk membuat lokasi default *working directory* pindah, kita dapat melakukannya dengan memilih pada menu: **Tools > Global options > pada “General” klik pada “Browse” dan pilih lokasi working directory yang diinginkan**. Proses tersebut ditampilkan pada Gambar 1.5

1.7 Memasang dan Mengaktifkan Paket R

`R` dapat ditingkatkan fungsionalitasnya melalui Paket-Paket yang tersedia secara luas. Paket-Paket ini dikembangkan secara spesifik oleh para pengembang sesuai dengan tujuan paketnya, seperti: `tidyverse` untuk *data science*, `pracma` untuk analisis diferensial, dll.

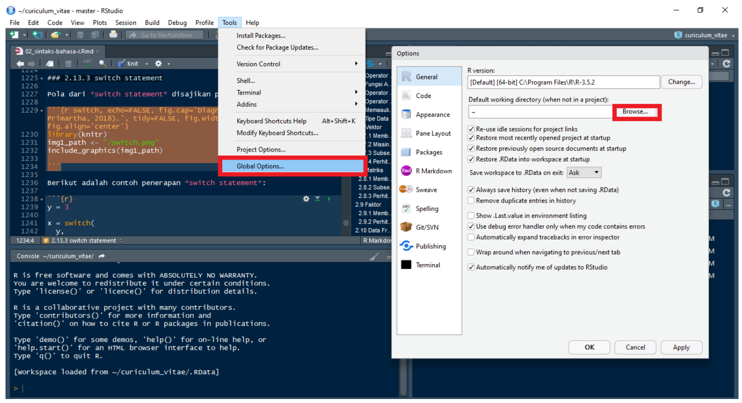

Figure 1.5: Merubah working directory melalui Global options.

Untuk menginstall Paket yang kita inginkan, kita dapat menggunakan fungsi `install.packages()`. Berikut adalah contoh bagaimana cara menginstall Paket `tidyverse`:

```
install.packages("tidyverse")
```

Paket yang telah diinstall tidak dapat langsung digunakan. Untuk menggunakan fungsi-fungsi yang tersedia pada Paket tersebut kita perlu terlebih dahulu mengaktifkannya menggunakan fungsi `library()`. Berikut adalah contoh sintaks untuk mengaktifkan Paket `tidyverse`:

```
library(tidyverse)
```

Bagaimana ingin menggunakan fungsi pada Paket namun tidak ingin mengaktifkan paketnya terlebih dahulu menggunakan fungsi `library()`? Untuk melakukannya kita perlu mengetikkan nama Paket dikuti oleh tanda "::" diikuti fungsi yang ingin kita gunakan. Berikut adalah contoh penggunaan fungsi `read_csv()` dari Paket `readr` (salah satu Paket yang terdapat pada kumpulan Paket `tidyverse`) untuk membaca file `contoh.csv`:

```
readr::read_csv("contoh.csv")
```

1.8 Fasilitas Help

Agar dapat menggunakan `R` dengan secara lebih baik, pengetahuan untuk mengakses fasilitas *help* in cukup penting untuk disampaikan. Adapun cara yang dapat digunakan adalah sebagai berikut.

1.8.1 Mencari Help dari Suatu Perintah Tertentu

Untuk memperoleh bantuan terkait suatu perintah tertentu kita dapat menggunakan fungsi `help()`. Secara umum format yang digunakan adalah sebagai berikut:

```
help(nama_perintah)
```

atau dapat juga menggunakan tanda tanya (?) pada awal `nama_perintah` seperti berikut:

```
?nama_perintah
```

Misalkan kita kebingungan terkait bagaimana cara menuliskan perintah untuk menghitung rata-rata suatu vektor. Kita dapat mengetikkan perintah berikut untuk mengakses fasilitas *help*.

```
help(mean)

#atau
?mean
```

Perintah tersebut akan memunculkan hasil berupa dokumentasi yang ditampilkan pada Gambar 1.6.

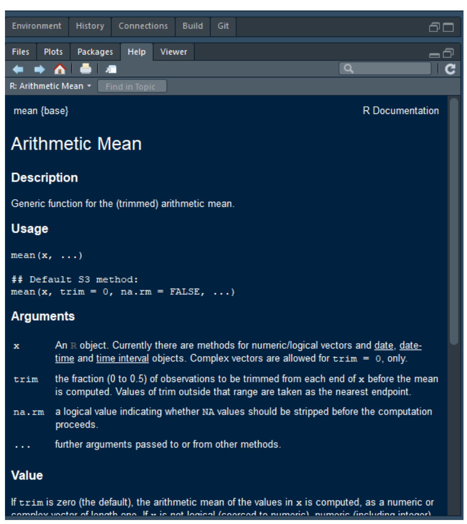

Figure 1.6: Jendela help dokumentasi fungsi mean().

Keterangan pada jendela pada Gambar 1.6 adalah sebagia berikut:

1. Pada bagian jendela kiri atas jendela *help*, diberikan keterangan nama dari perintah yang sedang ditampilkan.
2. Selanjutnya, pada bagian atas dokumen, ditampilkan infomasi terkait nama perintah, dan nama Paket yang memuat perintah tersebut. Pada gambar diatas informasi terkait perintah dan nama Paket ditunjukkan pada teks `mean {base}` yang menunjukkan perintah `mean()` pada Paket (Paket) *base* (Paket bawaan `R`).
3. Setiap jendela *help* dari suatu perintah tertentu selanjutnya akan memuat bagian-bagian berikut:

- *Title*
- *Description* : deskripsi singkat tentang perintah.
- *Usage* : menampilkan sintaks perintah untuk penggunaan perintah tersebut.
- *Arguments* : keterangan mengenai *argument/input*yang diperlukan pada perintah tersebut.
- *Details* : keterangan lebih lengkap lengkap tentang perintah tersebut.
- *Value* : keterangan tentang *output* suatu perintah dapat diperoleh pada bagian ini.
- *Author(s)* : memberikan keterangan tentang *Author* dari perintah tersebut.
- *References* : seringkali referensi yang dapat digunakan untuk memperoleh keterangan lebih lanjut terhadap suatu perintah ditampilkan pada bagian ini.
- *See also*: bagian ini berisikan daftar perintah/fungsi yang berhubungan erat dengan perintah tersebut.
- *Example* : berisikan contoh-contoh penggunaan perintah tersebut.

Kita juga dapat melihat contoh penggunaan dari perintah tersebut. Untuk melakukannya kita dapat menggunakan fungsi `example()`. Fungsi tersebut akan menampilkan contoh kode penerapan dari fungsi yang kita inginkan. Secara sederhana fungsi tersebut dapat dituliskan sebagai berikut:

```
example(nama_perintah)
```

Untuk mengetahui contoh kode fungsi `mean()`, ketikkan sintaks berikut:

```
example(mean)
```

```
##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50
```

kita juga dapat mencoba kode yang dihasilkan pada console `R`. Berikut adalah contoh penerapannya:

```
# Menghitung rata-rata bilangan 1 sampai 10 dan 50
# membuat vektor
x <- c(0:10, 50)

# Print
x
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 50
```

```
# mean
mean(x)
```

```
## [1] 8.75
```

Pembaca dapat mencoba melakukanya sendiri dengan mengganti nilai yang telah ada serta mencoba contoh kode yang lain.

1.8.2 General Help

Kita juga dapat membaca beberapa dokumen manual yang ada pada `R`. Untuk melakukannya jalankan perintah berikut:

```
help.start()
```

Output yang dihasilkan berupa link pada sejumlah dokumen yang dapat kita klik. Tampilan halaman yang dihasilkan disajikan pada Gambar 1.7.

1.8.3 Fasilitas Help Lainnya

Selain yang telah penulis sebutkan sebelumnya. Kita juga dapat memanfaatkan fasilitas *help* lainnya melalui fungsi `apropos()` dan `help.search()`.

`apropos ()`: mengembalikan daftar objek, berisi pola yang pembaca cari, dengan pencocokan sebagian. Ini berguna ketika pembaca tidak ingat persis nama fungsi yang akan digunakan. Berikut adalah contoh ketika penulis ingin mengetahui fungsi yang digunakan untuk menghitung median.

```
apropos("med")
```

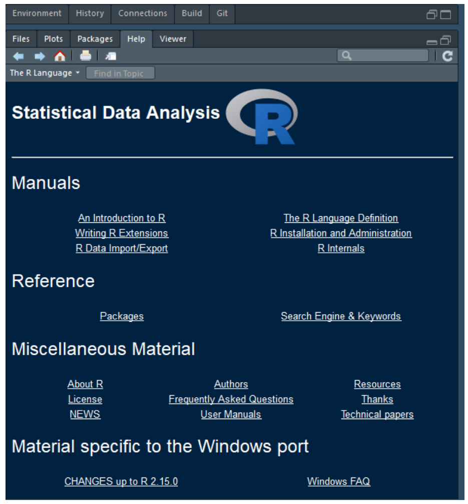

Figure 1.7: Jendela general help dokumentasi fungsi mean().

```
## [1] "elNamed"        "elNamed<-"      "median"
## [4] "median.default" "medpolish"      "runmed"
```

List yang dihasilkan berupa fungsi-fungsi yang memiliki elemen kata "med". Berdasarkan pencaria tersebut penulis dapat mencoba menggunakan fungsi "median" untuk menghitung median.

`help.search ()` (sebagai alternatif ??): mencari dokumentasi yang cocok dengan karakter yang diberikan dengan cara yang berbeda. Ini mengembalikan daftar fungsi yang mengandung istilah yang pembaca cari dengan deskripsi singkat dari fungsi.

Berikut adalah contoh penerapan dari fungsi tersebut:

```
help.search("mean")

# atau
??mean
```

Output yang dihasilkan akan tampak seperti pada Gambar 1.8.

1.9 Referensi

1. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung

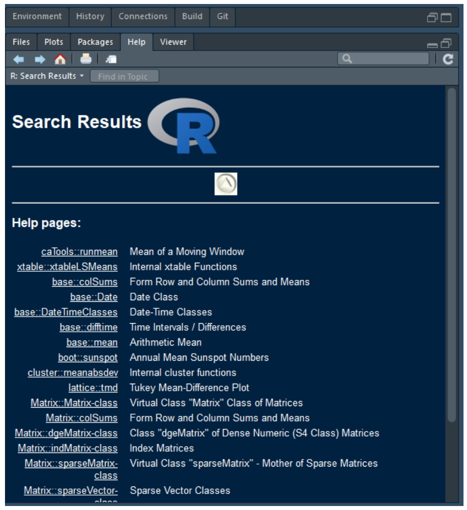

Figure 1.8: Jendela help search dokumentasi fungsi mean().

2. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta
3. STHDA. Running RStudio and Setting Up Your Working Directory - Easy R Programming .http://www.sthda.com/english/wiki/running-rstudio-and-setting-up-your-working-directory-easy-r-programming#set-your-working-directory
4. STDHA. **Getting Help With Functions In R Programming**. http://www.sthda.com/english/wiki/getting-help-with-functions-in-r-programming .
5. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.

Chapter 2

Kalkulasi Menggunakan R

Pada *Chapter* ini penulis akan menjelaskan bagaimana melakukan perhitungan menggunakan `R`. Hal-hal yang akan dibahas pada *chapter* ini antara lain:

- Operator dan fungsi dasar pada `R`
- Jenis dan struktur data
- Vektor (cara membuat dan melakukan operasi matematika pada vektor)
- Matriks (cara membuat dan melakukan operasi matematika pada matriks)

2.1 Operator Aritmatik

Proses perhitungan akan ditangani oleh fungsi khusus. `R` akan memahami urutannya secara benar. Kecuali kita secara eksplisit menetapkan yang lain. Sebagai contoh jalankan sintaks berikut:

```
2+4*2
```

```
## [1] 10
```

Bandingkan dengan sintaks berikut:

```
(2+4)*2
```

```
## [1] 12
```

> **Tips:** `R` dapat digunakan sebagai kalkulator

Berdasarkan kedua hasil tersebut dapat disimpulkan bahwa ketika kita tidak menetapkan urutan perhitungan menggunakan tanda kurung, `R` akan secara otomatis akan menghitung terlebih dahulu perkalian atau pembangian.

Operator aritmatika yang disediakan `R` disajikan pada Tabel 2.1:

Table 2.1: Operator Aritmatika `R`.

Simbol	**Keterangan**
+	*Addition*, untuk operasi penjumlahan
-	*Substraction*, untuk operasi pengurangan
*	*Multiplication*, untuk operasi pembagian
/	*Division*, untuk operasi pembagian
^	*Eksponentiation*, untuk operasi pemangkatan
%%	*Modulus*, Untuk mencari sisa pembagian
%/%	*Integer*, Untuk mencari bilangan bulat hasil pembagian saja dan tanpa sisa pembagian

Untuk lebih memahaminya berikut contoh sintaks penerapan operator tersebut.

```
# Addition
5+3
```

```
## [1] 8
```

```
# Substraction
5-3
```

```
## [1] 2
```

```
# Multiplication
5*3
```

```
## [1] 15
```

```
# Division
5/3
```

```
## [1] 1.667
```

```
# Eksponetiation
5^3
```

```
## [1] 125
```

```
# Modulus
5%%3
```

```
## [1] 2
```

```
# Integer
5%/%3
```

```
## [1] 1
```

> **Tips:** Pada `R` tanda `#` berfungsi menambahkan keterangan untuk menjelaskan sebuah sintaks pada `R`

2.2 Fungsi Aritmetik

Selain fungsi operator aritmetik, pada `R` juga telah tersedia fungsi aritmetik yang lain seperti logaritmik, ekponensial, trigonometri, dll.

1. Logaritma dan eksponensial

Untuk contoh fungsi logaritmik dan eksponensial jalankan sintaks berikut:

```
log2(8) # logaritma basis 2 untuk 8
```

```
## [1] 3
```

```
log10(8) # logaritma basis 10 untuk 8
```

```
## [1] 0.9031
```

```
exp(8) # eksponensial 8
```

```
## [1] 2981
```

2. Fungsi trigonometri

fungsi trigonometri yang ditampilkan seperti sin,cos, tan, dll.

```
cos(x) # cos x
sin(x) # Sin x
tan(x) # Tan x
acos(x) # arc-cos x
asin(x) # arc-sin x
atan(x) #arc-tan x
```

> **Penting!!!**
>
> x dalam fungsi trigonometri memiliki satuan radian

Berikut adalah salah satu contoh penggunaannya:

```
cos(pi)
```

```
## [1] -1
```

Pada Paket `pracma` fungsi-fungsi trigonometri dapat ditambah lagi. Fungsi-fungsi tersebut antara lain:

```
cot(x) # cotan x
csc(x) # cosecan x
sec(x) # secan x
acot(x) # arc-cotan x
acsc(x) # arc-cosecan x
asec(x) # arc-secan x
```

3. Fungsi Hiperbolik

fungsi hiperbolik yang tersedia antara lain:

```
cosh(x)
sinh(x)
tanh(x)
acosh(x)
asinh(x)
atanh(x)
```

Fungsi tersebut dapat ditambah lagi dari Paket `pracma`. Fungsi-fungsi yang tersedia antara lain:

```
coth(x)
csch(x)
sech(x)
acoth(x)
acsch(x)
asech(x)
```

4. Fungsi matematik lainnya

Fungsi lainnya yang dapat digunakan adalah fungsi absolut, akar kuadrat, dll. Berikut adalah contoh sintaks penggunaan fungsi absolut dan akar kuadrat.

```
abs(-2) # nilai absolut -2
```

```
## [1] 2
```

```
sqrt(4) # akar kuadrat 4
```

```
## [1] 2
```

2.3 Operator Relasi

Operator relasi digunakan untuk membandingkan satu objek dengan objek lainnya. Operator yang disediakan `R` disajikan pada Tabel 2.2.

Table 2.2: Operator Relasi `R`.

Simbol	**Keterangan**
">"	Lebih besar dari
"<"	Lebih Kecil dari
"=="	Sama dengan
">="	Lebih besar sama dengan
"<="	Lebih kecil sama dengan
"!="	Tidak sama dengan

Berikut adalah penerapan operator pada tabel tersebut:

```
x <- 34
y <- 35

# Operator >
x > y
```

```
## [1] FALSE
```

```
# Operator <
x < y
```

```
## [1] TRUE
```

```
# operator ==
x == y
```

```
## [1] FALSE
```

```
# Operator >=
x >= y
```

```
## [1] FALSE
```

```
# Operator <=
x <= y
```

```
## [1] TRUE
```

```
# Operator !=
x != y
```

```
## [1] TRUE
```

2.4 Operator Logika

Operator logika hanya berlaku pada vektor dengan tipe logical, numeric, atau complex. Semua angka bernilai 1 akan dianggap bernilai logika `TRUE`. Operator logika yang disediakan `R` dapat dilihat pada Tabel 2.3.

Table 2.3: Operator logika `R`.

Simbol	Keterangan
“&&” ”	Operator logika AND
“!”	Opeartor logika NOT

Simbol	Keterangan
"&" "	Operator logika AND element wise "

Penerapannya terdapat pada sintaks berikut:

```
v <- c(TRUE,TRUE, FALSE)
t <- c(FALSE,FALSE,FALSE)

# Operator !
print(!v)
```

```
## [1] FALSE FALSE  TRUE
```

```
# operator &
print(v&t)
```

```
## [1] FALSE FALSE FALSE
```

```
# Operator |
print(v|t)
```

```
## [1]  TRUE  TRUE FALSE
```

> **Penting!!!**
>
> - operator `&` dan `|` akan mengecek logika tiap elemen pada vektor secara berpesangan (sesuai urutan dari kiri ke kanan). Operator `%%` dan `||` hanya mengecek dari kiri ke kanan pada
> - observasi pertama. Misal saat menggunakan && jika observasi pertama `TRUE` maka observasi pertama pada vektor lainnya akan dicek, namun jika observasi pertama `FALSE` maka proses akan segera dihentikan dan menghasilkan FALSE.

2.5 Memasukkan Nilai Kedalam Variabel

Variabel pada `R` dapat digunakan untuk menyimpan nilai. Sebagai contoh jalankan sintaks berikut:

```r
# Harga sebuah lemon adalah 500 rupiah
lemon <- 500

# Atau
500 -> lemon

# dapat juga menggunakan tanda "="
lemon = 500
```

> **Penting!!!**
>
> 1. `R` memungkinkan penggunaan `<-`,`->`, atau `=` sebagai perintah pengisi nilai variabel
> 2. `R` bersifat *case-sensitive*. Maksudnya adalah variabel Lemon tidak sama dengan lemon (Besar kecil huruf berpengaruh)

Untuk mengetahui nilai dari objek `lemon` kita dapat menggunakan fungsi `print()` atau mengetikkan nama objeknya secara langsung.

```r
# Menggunakan fungsi print()
print(lemon)
```

```
## [1] 500
```

```r
# Atau
lemon
```

```
## [1] 500
```

`R` akan menyimpan variabel `lemon` sebagai objek pada memori. Sehingga kita dapat melakukan operasi terhadap objek tersebut seperti mengalikannya atau menjumlahkannya dengan bilangan lain. Sebagai contoh jalankan sintaks berikut:

```r
# Operasi perkalian terhadap objek lemon
5*lemon
```

```
## [1] 2500
```

Kita dapat juga mengubah nilai dari objek `lemon` dengan cara menginput nilai baru terhadap objek yang sama. `R` secara otomatis akan menggatikan nilai sebelumnya. Untuk lebih memahaminya jalankan sintaks berikut:

```
lemon <- 1000

# Print lemon
print(lemon)
```

```
## [1] 1000
```

Untuk lebih memahaminya berikut adalah sintaks untuk menghitung volume suatu objek.

```
# Dimensi objek
panjang <- 10
lebar <- 5
tinggi <- 5

# Menghitung volume
volume <- panjang*lebar*tinggi

# Print objek volume
print(volume)
```

```
## [1] 250
```

Untuk mengetahui objek apa saja yang telah kita buat sepanjang artikel ini kita dapang menggunakan fungsi `ls()`.

```
ls()
```

```
##  [1] "A"          "B"          "img1_path" "lebar"
##  [5] "lemon"      "panjang"    "t"          "tinggi"
##  [9] "v"          "volume"     "x"          "xm"
## [13] "y"
```

> **Catatan:** Kumpulan objek yang telah tersimpan dalam memori disebut sebagai **workspace**

Untuk menghapus objek pada memori kita dapat menggunakan fungsi `rm()`. Pada sintaks berikut penulis hendak menghapus objek `lemon` dan `volume`.

```
# Menghapus objek lemon dan volume
rm(lemon, volume)

# Tampilkan kembali objek yang tersisa
ls()
```

```
##  [1] "A"         "B"         "img1_path" "lebar"
##  [5] "panjang"   "t"         "tinggi"    "v"
##  [9] "x"         "xm"        "y"
```

> **Tips:** Setiap variabel atau objek yang dibuat akan menempati sejumlah memori pada komputer sehingga jika kita bekerja dengan jumlah data yang banyak pastikan kita menghapus seluruh objek pada memori sebelum memulai kerja.

2.6 Tipe dan Struktur Data

Data pada `R` dapat dikelompokan berdasarkan beberapa tipe. Tipe data pada `R` disajikan pada Tabel 2.4.

Table 2.4: Tipe data `R`.

Tipe Data	**Contoh**	**Keterangan**
Logical	TRUE, FALSE	Nilai Boolean
Numeric	12.3, 5, 999	Segala jenis angka
Integer	23L, 97L, 3L	Bilangan integer (bilangan bulat)
Complex	2i, 3i, 9i	Bilangan kompleks
Character	'a', "b", "123"	Karakter dan string
Factor	1, 0, "Merah"	Dapat berupa numerik atau string (namun pada proses akan terbaca sebagai angka)
Raw	Identik dengan "hello"	Segala jenis data yang disimpan sebagai raw bytes

Sintaks berikut adalah contoh dari tipe data pada `R`. Untuk mengetahui tipa data suatu objek kita dapat menggunakan perintah `class()`

```
# Logical
apel <- TRUE
class(apel)
```

```
## [1] "logical"
```

```
# Numeric
x <- 2.3
class(x)
```

```
## [1] "numeric"
```

```
# Integer
y <- 2L
class(y)
```

```
## [1] "integer"
```

```
# Compleks
z <- 5+2i
class(z)
```

```
## [1] "complex"
```

```
# string
w <- "saya"
class(w)
```

```
## [1] "character"
```

```
# Raw
xy <- charToRaw("hello world")
class(xy)
```

```
## [1] "raw"
```

Keenam jenis data tersebut disebut sebagai tipe data atomik. Hal ini disebabkan karena hanya dapat menangani satu tipe data saja. Misalnya hanya numeric atau hanya integer.

Selain menggunakan fungsi `class()`, kita dapat pula menggunakan fungsi `is_numeric()`, `is.character()`, `is.logical()`, dan sebagainya berdasarkan jenis data apa yang ingin kita cek. Berbeda dengan fungsi `class()`, ouput yang dihasilkan pada fungsi seperti `is_numeric()` adalah nilai Boolean sehingga fungsi ini hanya digunakan untuk mengecek apakah jenis data pada objek sama seperti yang kita pikirkan. Sebagai contoh disajikan pada sintaks berikut:

```
data <- 25

# Cek apakah objek berisi data numerik
is.numeric(data)
```

```
## [1] TRUE
```

```
# Cek apakah objek adalah karakter
is.character(data)
```

```
## [1] FALSE
```

Kita juga dapat mengubah jenis data menjadi jenis lainnya seperti integer menjadi numerik atau sebaliknya. Fungsi yang digunakan adalah `as.numeric()` jika ingin mengubah suatu jenis data menjadi numerik. Fungsi lainnya juga dapat digunakan sesuai dengan kita ingin mengubah jenis data objek menjadi jenis data lainnya.

```
# Integer
apel <- 2L

# Ubah menjadi numerik
as.numeric(apel)
```

```
## [1] 2
```

```
# Cek
is.numeric(apel)
```

```
## [1] TRUE
```

```
# Logical
nangka <- TRUE

# Ubah logical menjadi numeric
as.numeric(nangka)
```

```
## [1] 1
```

```
# Karakter
minum <- "minum"

# ubah karakter menjadi numerik
as.numeric(minum)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA
```

> **Penting!!!**
>
> Konversi karakter menjadi numerik akan menghasilkan output NA (*not available*). `R` tidak mengetahui bagaimana cara merubah karakter menjadi bentuk numerik.

Berdasarkan Tabel 2, vektor karakter dapat dibuat menggunakan tanda kurung baik *double quote* (" ") maupun *single quote* (' ').Jika pada teks yang kita tuliskan mengandung *quote* maka kita harus menghentikannya menggunakan tanda ( ). Sbegai contoh kita ingin menuliskan '**My friend's name is "Adi"**, pada sintaks akan dituliskan:

```
'My friend\`s name is "Adi"'
```

```
## [1] "My friend`s name is \"Adi\""
```

```
# Atau

"My friend's name \"Adi\""
```

```
## [1] "My friend's name \"Adi\""
```

Struktur data diklasifikasikan berdasarkan dimensi data dan tie data di dalamnya (homogen atau heterogen). Klasifikasi jenis data disajikan pada Tabel 2.5.

Table 2.5: Struktur data `R`.

Dimensi	**Homogen**	**Heterogen**
1d	Atomik vektor	List
2d	Matriks	Dataframe
nd	Array	

Berdasarkan Tabel tersebut dapat kita lihat bahwa objek terbagi atas dua buah struktur data yaitu homogen dan heterogen. Objek dengan struktur data homogen hanya dapat menyimpan satu tipe atau jenis data saja (numerik saja atau factor saja), sedangkan objek dengan struktur data heterogen akan dapat menyimpan berbagai jenis data.

2.7 Vektor

Vektor merupakan kombinasi berbagai nilai (numerik, karakter, logical, dan sebagainya berdasarkan jenis input data) pada objek yang sma. Pada contoh kasus berikut, pembaca akan memiliki sesuai jenis data input yaitu**vektor numerik**, **vector karakter**, **vektor logical**, dll.

2.7.1 Membuat vektor

Vektor dibuat dengan menggunakan fungsi `c()`(concatenate) seperti yang disajikan pada sintaks berikut:

```
# membuat vektor numerik
x <- c(3,3.5,4,7)
x # print vektor
```

```
## [1] 3.0 3.5 4.0 7.0
```

```
# membuat vektor karakter
y <- c("Apel", "Jeruk", "Rambutan", "Salak")
y # print vektor
```

```
## [1] "Apel"     "Jeruk"    "Rambutan" "Salak"
```

```
# membuat vektor logical
t <- c("TRUE", "FALSE", "TRUE")
t # print vektor
```

```
## [1] "TRUE"  "FALSE" "TRUE"
```

selain menginput nilai pada vektor, kita juga dapat memberi nama nilai setiap vektor menggunakan fungsi `names()`.

```
# Membuat vektor jumlah buah yang dibeli
Jumlah <- c(5,5,6,7)
names(Jumlah) <- c("Apel", "Jeruk", "Rambutan", "Salak")

# Atau
Jumlah <- c(Apel=5, Jeruk=5, Rambutan=6, Salak=7)

# Print
Jumlah
```

```
##     Apel    Jeruk Rambutan    Salak
##        5        5        6        7
```

> **Penting!!!**
>
> Vektor hanya dapat memuat satu buah jenis data. Vektor hanya dapat mengandung jenis data numerik saja, karakter saja, dll.

Untuk menentukan panjang sebuah vektor kita dapat menggunakan fungsi `lenght()`.

```
length(Jumlah)
```

```
## [1] 4
```

2.7.2 Missing Values

Seringkali nilai pada vektor kita tidak lengkap atau terdapat nilai yang hilang (*missing value*) pada vektor. *Missing value* pada `R` dilambangkan oleh `NA`(*not available*). Berikut adalah contoh vektor dengan *missing value*.

```
Jumlah <- c(Apel=5, Jeruk=NA, Rambutan=6, Salak=7)
```

Untuk mengecek apakah dalam objek terdapat *missing value* dapat menggunakan fungsi `is.na()`. ouput dari fungsi tersebut adalah nilai Boolean. Jika terdapat *Missing value*, maka output yang dihasilkan akan memberikan nilai `TRUE`.

```
is.na(Jumlah)
```

```
##     Apel    Jeruk Rambutan    Salak
##    FALSE     TRUE    FALSE    FALSE
```

Penting!!!

1. Selain `NA` terdapat NaN (*not a number*) sebagai *missing value8*. Nilai tersebut muncul ketika fungsi matematika yang digunakan pada proses perhitungan tidak bekerja sebagaimana mestinya. Contoh: `0/0 = NaN`
2. `is.na()` juga akan menghasilkan nilai `TRUE` pada NaN. Untuk membedakannya dengan `NA` dapat digunakan fungsi `is.nan()`.

2.7.3 Subset Pada Vektor

Subseting vector terdiri atas tiga jenis, yaitu: *positive indexing*, *Negative Indexing*, dan .

- **Positive indexing**: memilih elemen vektor berdasarkan posisinya (indeks) dalam kurung siku.

```r
# Subset vektor pada urutan kedua
Jumlah[2]
```

```
## Jeruk
##    NA
```

```r
# Subset vektor pada urutan 2 dan 4
Jumlah[c(2, 4)]
```

```
## Jeruk Salak
##    NA     7
```

Selain melalui urutan (indeks), kita juga dapat melakukan subset (membuat himpunan bagian) berdasarkan nama elemen vektornya.

```r
Jumlah["Jeruk"]
```

```
## Jeruk
##    NA
```

> **Penting!!!**
>
> Indeks pada `R` dimulai dari 1. Sehingga kolom atau elemen pertama vektor dimulai dari [1]

- **Negative indexing**: mengecualikan (*exclude*) elemen vektor.

```r
# mengecualikan elemen vektor 2 dan 4
Jumlah[-c(2,4)]
```

```
##     Apel Rambutan
##        5        6
```

```r
# mengecualikan elemen vektor 1 sampai 3
Jumlah[-c(1:3)]
```

```
## Salak
##     7
```

- **Subset berdasarkan vektor logical**: Hanya, elemen-elemen yang nilai yang bersesuaian dalam vektor pemilihan bernilai TRUE, akan disimpan dalam subset.

Penting!!!

panjang vektor yang digunakan untuk subset harus sama.

```
Jumlah <- c(Apel=5, Jeruk=NA, Rambutan=6, Salak=7)

# selecting vector
merah <- c(TRUE, FALSE, TRUE, FALSE)

# Subset
Jumlah[merah==TRUE]
```

```
##     Apel Rambutan
##        5        6
```

```
# Subset untuk elemen vektor bukan missing value
Jumlah[!is.na(Jumlah)]
```

```
##     Apel Rambutan    Salak
##        5        6        7
```

2.7.4 Operasi Matematis Menggunakan Vektor

Jika pembaca melakukan operasi dengan vektor, operasi akan diterapkan ke setiap elemen vektor. Contoh disediakan pada sintaks di bawah ini:

```
pendapatan <- c(2000, 1800, 2500, 3000)
names(pendapatan) <- c("Andi", "Joni", "Lina", "Rani")
pendapatan
```

```
## Andi Joni Lina Rani
## 2000 1800 2500 3000
```

```
# Kalikan pendapatan dengan 3
pendapatan*3
```

```
## Andi Joni Lina Rani
## 6000 5400 7500 9000
```

Seperti yang dapat dilihat, `R` mengalikan setiap elemen dengan bilangan pengali.

Kita juga dapat mengalikan vektor dengan vektor lainnya.Contohnya disajikan pada sintaks berikut:

```
# membuat vektor dengan panjang
# sama dengan dengan vektor pendapatan
coefs <- c(2, 1.5, 1, 3)

# Mengalikan pendapatan dengan vektor coefs
pendapatan*coefs
```

```
## Andi Joni Lina Rani
## 4000 2700 2500 9000
```

Berdasarkan sintaks tersebut dapat terlihat bahwa operasi matematik terhadap masing-masing vektor dapat berlangsung jika panjang vektornya sama.

Berikut adalah fungsi lain yang dapat digunakan pada operasi matematika vektor.

```
max(x) # memperoleh nilai maksimum x
min(x) # memperoleh nilai minimum x
range(x) # memperoleh range vektor x
length(x) # memperoleh jumlah vektor x
sum(x) # memperoleh total penjumlahan vektor x
prod(x) # memeperoleh produk elemen vektor x
mean(x) # memperoleh nilai mean vektor x
sd(x) # standar deviasi vektor x
var(x) # varian vektor x
sort(x) # mengurutkan elemen vektor x dari yang terbesar
```

Contoh penggunaan fungsi tersebut disajikan beberapa pada sintaks berikut:

```
# Menghitung range pendapatan
range(pendapatan)
```

```
## [1] 1800 3000
```

```
# menghitung rata-rata dan standar deviasi pendapatan
mean(pendapatan)
```

```
## [1] 2325
```

```
sd(pendapatan)
```

```
## [1] 537.7
```

2.7.5 Membuat Deret Angka

Secara sederhana vektor merupakan deret angka. Vektor bisa jadi berupa data yang kita miliki atau sengaja kita buat untuk tujuan simulasi matematika. Urutan angka-angka ini bisa memiliki interval konstan, contoh: titik waktu pada analisis reaksi kimia, atau dapat pula intervalnya bersifat acak seperti pada simulasi Monte Carlo.

2.7.5.1 *Regular Sequences*

Operator *colon* (":") dapat digunakan untuk membuat *sequence vector*. Operator tersebut berfungsi sebagai pemisah antara nilai awal dan akhir deret bilangan. Interval nilai *sequence* yang terbentuk adalah '. Berikut adalah contoh bagaimana cara membuat *sequence vector* menggunakan operator *colon*:

```
# vektor benilai 1 s/d 10
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# vektor bernilai 10 s/d -1
10:-1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1  0 -1
```

Perlu diperhatikan bahwa dalam aplikasinya operator *colon* memiliki prioritas tinggi untuk dilakukan komputasi terlebih dahulu dibandingkan operator matematika. Perhatikan sintaks berikut:

```
n = 10

# membuat vektor bernilai 0 s/d 9
1:n-1
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
# membuat vektor bernilai 1 s/d 9
1:(n-1)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

Jika kita menginginkan interval antar angka selain 1, kita dapat menggunakan fungsi `seq()`. Format sintaks tersebut adalah sebagai berikut:

```
seq(from, to, by)
```

> **Catatan:**
>
> - **from, to**: angka awal dan akhir atau nilai maksimum dan minimum deret bilangan yang diinginkan.
> - **by**: interval antar nilai

Misalkan kita akan membuat deret bilangan dari 3 sampai 8 dengan interval antar deret sebesar 0,5. Berikut adalah sintaks yang digunakan:

```
seq(from=3,to=8,by=0.5)
```

```
##  [1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
```

2.7.6 Nilai Berulang

Fungsi `rep()` dapat digunakan untuk membuat deret dengan nilai berulang. Format fungsi tersebut adalah sebagai berikut:

```
rep(x, times, each)
```

> **Catatan:**
>
> - **x**: nilai yang hendak dibuat berulang.
> - **times**: jumlah pengulangan.
> - **each**: argumen tambahan yang menentukan jumlah masing-masing elemen vektor akan dicetak.

```
# cetak angka 5 sebanyak 5 kali
rep(x=5, times=5)
```

```
## [1] 5 5 5 5 5
```

```
# cetak angka 5 dan 6 sebanyak 3 kali
rep(c(5,6), times=3)
```

```
## [1] 5 6 5 6 5 6
```

```
# cetak angka 5 dan 6 masing-masing 3 kali
rep(c(5,6), each=3)
```

```
## [1] 5 5 5 6 6 6
```

2.7.7 Deret Bilangan Acak

Deret bilangan acak biasanya banyak digunakan dalam sebuah simulasi. `R` menyediakan fungsi untuk memproduksi bilangan-bilangan acak tersebut berdasarkan distribusi tertentu. Berikut adalah tabel rangkuman nama distribusi, fungsi, dan argumen yang digunakan:

Table 2.6: Ringkasan Fungsi dan Argumen Distribusi Probabilitas.

Distribusi	Fungsi	Argumen
Beta	`rbeta(n, shape1, shape2, ncp = 0)`	`n` = jumlah observasi; `shape1,shape2` = parameter non-negatif distribusi beta; `ncp` = *non-centrality parameter*
Binomial	`rbinom(n, size, prob)`	`n`= jumlah observasi; `prob` = probabilitas sukses; `size` = jumlah percobaan
Cauchy	`rcauchy(n, location = 0, scale = 1)`	`n` = jumlah observasi; `location, scale` = parameter lokasi dan skala distribusi Cauchy
Chi-Square	`rchisq(n, df, ncp = 0)`	`n` = jumlah observasi; `df` = derajat kebebasan; `ncp` = *non-centrality parameter*
Exponensial	`rexp(n, rate = 1)`	`n` = jumlah observasi; `rate` = vektor parameter *rate*
F	`rf(n, df1, df2, ncp)`	`n` = jumlah observasi; `df1, df2` = derajat kebebasan; `ncp` = *non-centrality parameter*
Gamma	`rgamma(n, shape, rate = 1, scale = 1/rate)`	`n` = jumlah observasi; `shape, scale` = parameter *shape* dan *scale*; `rate` = alternatif lain argumen `rate`
Geometrik	`rgeom(n, prob)`	`n` = jumlah observasi; `prob` = probabilitas sukses
Hipergeometrik	`rhyper(nn, m, n, k)`	`nn` = jumlah observasi; `m` = jumlah bola putih dalam wadah; `n` = jumlah bola hitam dalam wadah; `k` = jumlah pengambilan
Log-normal	`rlnorm(n, meanlog = 0, sdlog = 1)`	`n` = jumlah observasi; `meanlog, sdlog` = nilai mean dan simpangan baku dalam skala logaritmik

Distribusi	Fungsi	Argumen
Negatif Binomial	`rnbinom(n, size, prob, mu)`	`n` = jumlah observasi; `size` = target jumlah percobaan sukses pertama kali; `prob` = probabilitas sukses; `mu` = parameterisasi alternatif melalui mean
Normal	`rnorm(n, mean = 0, sd = 1)`	`n` = jumlah observasi; `mead, sd` = nilai mean dan simpangan baku
Poisson	`rpois(n, lambda)`	`n` = jumlah observasi; `lambda` = vektor nilai mean
Student t	`rt(n, df, ncp)`	`n` = jumlah observasi; `df` = derajat kebebasan; `ncp` = *non-centrality parameter*
Uniform	`runif(n, min = 0, max = 1)`	`n` = jumlah observasi; `min, max` = nilai maksimum dan minimum distribusi
Weibull	`rweibull(n, shape, scale = 1)`	`n` = jumlah observasi; `shape, scale` = parameter *shape* dan *scale*

Berikut adalah contoh pembuatan vektor menggunakan bilangan acak berdistribusi normal:

```
x <- 1:6
error <- rnorm(n=1, mean=0, sd=1)

# cetak x + error dengan 3 nilai signifikan
round((x+error), 3)
```

```
## [1] 1.205 2.205 3.205 4.205 5.205 6.205
```

2.8 Matriks

Matriks seperti Excel sheet yang berisi banyak baris dan kolom (kumpulan berapa vektor). Matriks digunakan untuk menggabungkan vektor dengan tipe yang sama, yang bisa berupa numerik, karakter, atau logis. Matriks digunakan untuk menyimpan tabel data dalam R. Baris-baris matriks pada umumnya adalah individu / pengamatan dan kolom adalah variabel.

2.8.1 Membuat matriks

Untuk membuat matriks kita dapat menggunakan fungsi `cbind()` atau `rbind()`. Berikut adalah contoh sintaks untuk membuat matriks.

```r
# membuat vektor numerik
col1 <- c(5, 6, 7, 8, 9)
col2 <- c(2, 4, 5, 9, 8)
col3 <- c(7, 3, 4, 8, 7)

# menggabungkan vektor berdasarkan kolom
my_data <- cbind(col1, col2, col3)
my_data
```

```
##      col1 col2 col3
## [1,]    5    2    7
## [2,]    6    4    3
## [3,]    7    5    4
## [4,]    8    9    8
## [5,]    9    8    7
```

```r
# Mengubah atau menambahkan nama baris
rownames(my_data) <- c("row1", "row2",
                       "row3", "row4",
                       "row5")
my_data
```

```
##      col1 col2 col3
## row1    5    2    7
## row2    6    4    3
## row3    7    5    4
## row4    8    9    8
## row5    9    8    7
```

> **Catatan:**
>
> - `cbind()`: menggabungkan objek `R` berdasarkan kolom
> - `rbind()`: menggabungkan objek `R` berdasarkan baris
> - `rownames()`: mengambil atau menetapkan nama-nama baris dari objek seperti-matriks
> - `colnames()`: mengambil atau menetapkan nama-nama kolom dari objek seperti-matriks

````
Kita dapat melakukan tranpose (merotasi matriks sehingga kolom menjadi baris dan sebaliknya) meng

```r
t(my_data)
````
````

```
##      row1 row2 row3 row4 row5
## col1    5    6    7    8    9
## col2    2    4    5    9    8
## col3    7    3    4    8    7
```

Selain melalui pembentukan sejumlah objek vektor, kita juga dapat membuat matriks menggunakan fungsi `matrix()`. Secara sederhana fungsi tersebut dapat dituliskan sebagai berikut:

```r
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

Catatan:

- `data`: vektor data opsional
- `nrow`, **ncol**: jumlah baris dan kolom yang diinginkan, masing-masing.
- `byrow`: nilai logis. Jika FALSE (default) matriks diisi oleh kolom, jika tidak, matriks diisi oleh baris.
- `dimnames`: Daftar dua vektor yang memberikan nama baris dan kolom masing-masing.

```
Dalam kode `R` di bawah ini, data input memiliki panjang 6. Kita ingin membuat matriks


```r
data <- matrix(
           data = c(1,2,3, 11,12,13),
           nrow = 2, byrow = TRUE,
           dimnames = list(c("row1", "row2"),
                           c("C.1", "C.2", "C.3"))
           )
data
```

```
##      C.1 C.2 C.3
## row1   1   2   3
## row2  11  12  13
```

Untuk mengetahui dimensi dari suatu matriks, kita dapat menggunakan fungsi `ncol()` untuk mengetahui jumlah kolom matriks dan `nrow()` untuk mengetahui jumlah baris pada matriks. Berikut adalah contoh penerapannya:
```

```
# mengetahui jumlah kolom
ncol(my_data)
```

```
## [1] 3
```

```
# mengetahui jumlah baris
nrow(my_data)
```

```
## [1] 5
```

Jika ingin memperoleh ringkasan terkait dimensi matriks kita juga dapat mengunakan fungsi `dim()` untuk mengetahui jumlah baris dan kolom matriks. Berikut adalah contoh penerapannya:

```
dim(my_data) # jumlah baris dan kolom
```

```
## [1] 5 3
```

2.8.2 Subset Pada Matriks

Sama dengan vektor, subset juga dapat dilakukan pada matriks. Bedanya subset dilakukan berdasarkan baris dan kolom pada matriks.

- **Memilih baris/kolom** berdasarkan pengindeksan positif

baris atau kolom dapat diseleksi menggunakan format `data[row, col]`. Cara selesi ini sama dengan vektor, bedanya kita harus menetukan baris dan kolom dari data yang akan kita pilih. Berikut adalah contoh penerapannya:

```
# Pilih baris ke-2
my_data[2,]
```

```
## col1 col2 col3
##    6    4    3
```

```
# Pilih baris 2 sampai 4
my_data[2:4,]
```

```
##      col1 col2 col3
## row2    6    4    3
## row3    7    5    4
## row4    8    9    8
```

```
# Pilih baris 2 dan 4
my_data[c(2,4),]
```

```
##      col1 col2 col3
## row2    6    4    3
## row4    8    9    8
```

```
# Pilih baris 2 dan kolom 3
my_data[2, 3]
```

```
## [1] 3
```

- **Pilih berdasarkan nama baris/kolom**

Berikut adalah contoh subset berdasarkan nama baris atau kolom.

```
# Pilih baris 1 dan kolom 3
my_data["row1","col3"]
```

```
## [1] 7
```

```
# Pilih baris 1 sampai 4 dan kolom 3
baris <- c("row1","row2","row3")
my_data[baris, "col3"]
```

```
## row1 row2 row3
##    7    3    4
```

- **Kecualikan baris/kolom** dengan pengindeksan negatif

Sama seperti vektor pengecualian data dapat dilakukan di matriks menggunakan pengindeksan negatif. Berikut cara melakukannya:

```
# Kecualikan baris 2 dan 3 serta kolom 3
my_data[-c(2,3), -3]
```

```
##      col1 col2
## row1    5    2
## row4    8    9
## row5    9    8
```

- **Pilihan dengan logik**

Dalam kode `R` di bawah ini, misalkan kita ingin hanya menyimpan baris di mana col3$>=4$:

```
col3 <- my_data[, "col3"]
my_data[col3 >= 4, ]
```

```
##      col1 col2 col3
## row1    5    2    7
## row3    7    5    4
## row4    8    9    8
## row5    9    8    7
```

2.8.3 Perhitungan Menggunakan Matriks

__ Kita juga dapat melakukan operasi matematika pada matriks. Pada operasi matematika pada matriks proses yang terjadi bisa lebih kompleks dibanding pada vektor, dimana kita dapat melakukan operasi untuk memperoleh gambaran data pada tiap kolom atau baris.

Berikut adalah contoh operasi matematika sederhana pada matriks:

```
# mengalikan masing-masing elemen matriks dengan 2
my_data*2
```

```
##      col1 col2 col3
## row1   10    4   14
## row2   12    8    6
## row3   14   10    8
## row4   16   18   16
## row5   18   16   14
```

```
# memperoleh nilai log basis 2 pada masing-masing elemen matriks
log2(my_data)
```

```
##       col1  col2  col3
## row1 2.322 1.000 2.807
## row2 2.585 2.000 1.585
## row3 2.807 2.322 2.000
## row4 3.000 3.170 3.000
## row5 3.170 3.000 2.807
```

Seperti yang telah penulis jelaskan sebelumnya, kita juga dapat melakukan operasi matematika untuk memperoleh hasil penjumlahan elemen pada tiap baris atau kolom dengan menggunakan fungsi `rowSums()` untuk baris dan `colSums()` untuk kolom.

```
# Total pada tiap kolom
colSums(my_data)
```

```
## col1 col2 col3
##   35   28   29
```

```
# Total pada tiap baris
rowSums(my_data)
```

```
## row1 row2 row3 row4 row5
##   14   13   16   25   24
```

Jika kita tertarik untuk mencari nilai rata-rata tiap baris arau kolom kita juga dapat menggunakan fungsi `rowMeans()` atau `colMeans()`. Berikut adalah contoh penerapannya:

```
# Rata-rata tiap baris
rowMeans(my_data)
```

```
##  row1  row2  row3  row4  row5
## 4.667 4.333 5.333 8.333 8.000
```

```
# Rata-rata tiap kolom
colMeans(my_data)
```

```
## col1 col2 col3
##  7.0  5.6  5.8
```

Kita juga dapat melakukan perhitungan statistika lainnya menggunakan fungsi `apply()`. Berikut adalah format sederhananya:

```
apply(x, MARGIN, FUN)
```

> **Catatan:**
>
> - `x` : data matriks

- `MARGIN` : Nilai yang dapat digunakan adalah `1` (untuk operasi pada baris) dan `2` (untuk operasi pada kolom)
- `FUN` : fungsi yang diterapkan pada baris atau kolom

untuk mengetahui fungsi (`FUN`) apa saja yang dapat diterapkan pada fungsi `apply()` jalankan sintaks bantuan berikut:

```
help(apply)
```

Berikut adalah contoh penerapannya:

```
# Rata-rata pada tiap baris
apply(my_data, 1, mean)
```

```
##  row1  row2  row3  row4  row5
## 4.667 4.333 5.333 8.333 8.000
```

```
# Median pada tiap kolom
apply(my_data, 2, median)
```

```
## col1 col2 col3
##    7    5    7
```

Perhitungan lainnya tidak akan dibahas pada *chapter* ini. Operasi matriks lebih lengkap selanjutnya akan dibahas pada *chapter* selanjutnya.

2.9 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
2. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
3. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.
4. STHDA. **Easy R Programming Basics**. http://www.sthda.com/english/wiki/easy-r-programming-basics
5. The R Core Team. 2018. **R: A Language and Environment for Statistical Computing**. R Manuals.
6. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.

Chapter 3

Visualisasi Data

Visualisasi data merupakan bagian yang sangat penting untuk mengkomunikasikan hasil analisa yang telah kita lakukan. Selain itu, komunikasi juga membantu kita untuk memperoleh gambaran terkait data selama proses analisa data sehingga membantu kita dalam memutuskan metode analisa apa yang dapat kita terapkan pada data tersebut.

`R` memiliki library visualisasi yang sangat beragam, baik yang merupakan fungsi dasar pada `R` maupun dari sumber lain seperti ggplot dan lattice. Seluruh library visualisasi tersebut memiliki kelebihan dan kekurangannya masing-masing.

Pada *chapter* ini kita tidak akan membahas seluruh library tersebut. Kita akab berfokus pada fungsi visualisasi dasar bawaan dari `R`. kita akan mempelajari mengenai jenis visualisasi data sampai dengan melakukan kustomisasi pada parameter grafik yang kita buat.

3.1 Visualisasi Data Menggunakan Fungsi plot()

Fungsi `plot()` merupakan fungsi umum yang digunakan untuk membuat plot pada `R`. Format dasarnya adalah sebagai berikut:

```
plot(x, y, type="p")
```

> **Catatan:**
>
> - **x dan y**: titik koordinat plot Berupa variabel dengan panjang atau jumlah observasi yang sama.
> - **type**: jenis grafik yang hendak dibuat. Nilai yang dapat dimasukkan antara lain:

- type="p" : membuat plot titik atau scatterplot. Nilai ini merupakan default pada fungsi `plot()`.
- type="l" : membuat plot garis.
- type="b" : membuat plot titik yang terhubung dengan garis.
- type="o" : membuat plot titik yang ditimpa oleh garis.
- type="h" : membuat plot garis vertikal dari titik ke garis y=0.
- type="s" : membuat fungsi tangga.
- type="n" : tidak membuat grafik plot sama sekali, kecuali plot dari axis. Dapat digunakan untuk mengatur tampilan suatu plot utama yang diikuti oleh sekelompok plot tambahan.

Untuk lebih memahaminya berikut penulis akan sajikan contoh untuk masing-masing grafik tersebut. Berikut adalah contoh sintaks dan hasil plot yang disajikan pada Gambar 3.1:

```
# membuat vektor data
x <- c(1:10); y <- x^2
```

```
# membagi jendela grafik menajdi 2 baris dan 4 kolom
par(mfrow=c(2,4))

# loop
type <- c("p","l","b","o","h","s","n")
for (i in type){
  plot(x,y, type= i,
       main= paste("type=", i))
}
```

Pada contoh selanjutnya kita akan mencoba membuat kembali data yang akan kita plotkan. Data pada contoh kali ini merupakan data suatu fungsi matematika. Berikut adalah sintaks yang digunakan:

```
set.seed(123)
x <- seq(from=0, to=10, by=0.1)
y <- x^2*exp(-x/2)*(1+rnorm(n=length(x), mean=0, sd=0.05))
```

```
par(mfrow=c(1,2),
    # mengatur margin grafik
    mar=c(4,4,1.5,1.5),
    # mengatur margin sumbu
    mex=0.8,
```

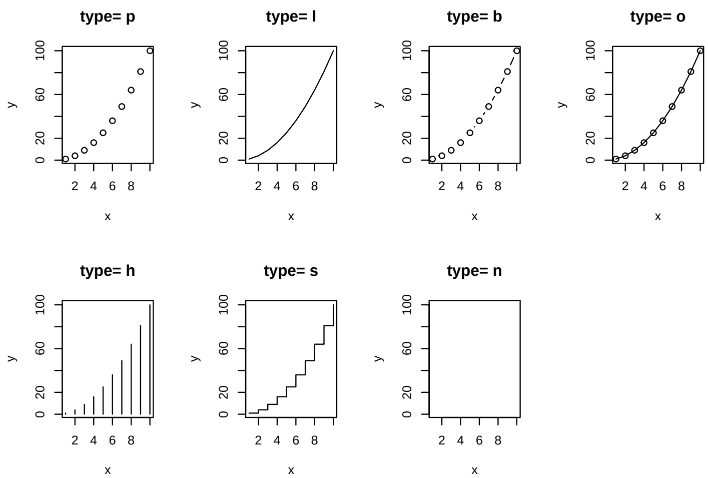

Figure 3.1: Plot berbagai jenis setting type

```
     # arah tick sumbu koordinat
     tcl=0.3)
plot(x, y, type="l")
plot(x, y, type="o")
```

Fungsi lain yang dapat digunakan untuk membuat kurva suatu persamaan matematis adalah fungsi `curve()`. Berbeda dengan fungsi `plot()` yang perlu menspesifikasi objek pada sumbu x dan y, fungsi `curve()` hanya perlu menspesifikasi objek sumbu x saja. Format fungsi `curve()` adalah sebagai berikut:

```
curve(expr, from = NULL, to = NULL, add = FALSE)
```

Catatan:

- **expr**: persamaan matematika
- **from dan to**: nilai awal dan akhir (maksimum atau minimum)
- **add**: nilai logik yang menentukan apakah kurva perlu ditambahkan kedalam kurva sebelumnya.

Berikut adalah contoh visualisasi menggunakan fungsi `curve()`:

```
par(mfrow=c(1,2),
     # mengatur margin grafik
     mar=c(4,4,1.5,1.5),
```

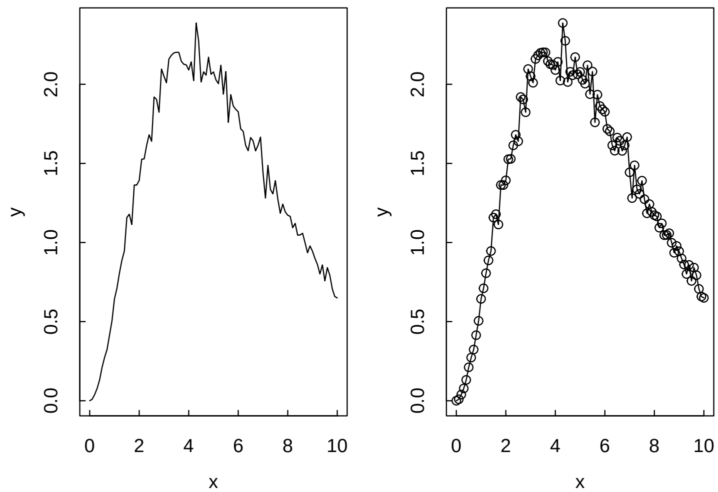

Figure 3.2: Plot fungsi matematika

```
    # mengatur margin sumbu
    mex=0.8,
    # arah tick sumbu koordinat
    tcl=0.3)

# Grafik kiri
curve(expr=x^2*exp(-x/2),
      from=0, to=10)

# Grafik kanan
plot(x, y, pch=19, cex=0.7,
     xlab="Waktu (detik)",
     ylab="Sinyal Intensitas")
curve(expr=x^2*exp(-x/2),
      from=0, to=10, add=TRUE)
```

3.2 Visualisasi Lainnya

Visualisasi lainnya yang sering digunakan antara lain: histogram, density plot, bar plot, dan box plot.

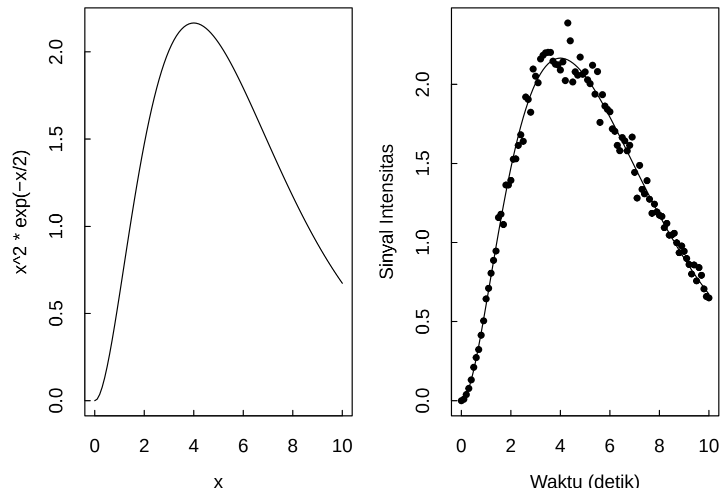

Figure 3.3: Visualisasi menggunakan fungsi curve (sebelah kiri) dan visualisasi menggunakan fungsi plot dan curve (sebelah kanan)

3.2.1 Bar Plot

Barplot pada `R` dapat dibuat menggunakan fungsi `barplot()`. Untuk lebih memahaminya berikut disajikan contoh barplot menggunakan dataset `VADeaths`. Untuk memuatnya jalankan sintaks berikut:

```
VADeaths
```

```
##       Rural Male Rural Female Urban Male Urban Female
## 50-54       11.7          8.7       15.4          8.4
## 55-59       18.1         11.7       24.3         13.6
## 60-64       26.9         20.3       37.0         19.3
## 65-69       41.0         30.9       54.6         35.1
## 70-74       66.0         54.3       71.1         50.0
```

Contoh bar plot untuk variabel `Rural Male` disajikan pada Gambar 3.4:

```
par(mfrow=c(1,2))
barplot(VADeaths[, "Rural Male"], main="a")
barplot(VADeaths[, "Rural Male"], main="b", horiz=TRUE)
```

```
par(mfrow=c(1,1))
```

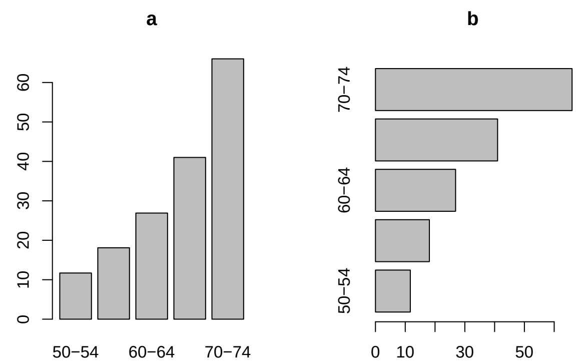

Figure 3.4: a. bar plot vertikal; b. bar plot horizontal

Kita dapat mengubah warna pada masing-masing bar, baik outline bar maupun box pada bar. Selain itu kita juga dapat mengubah nama grup yang telah dihasilkan sebelumnya. Berikut sintaks untuk melakukannya dan output yang dihasilkan pada Gambar 3.5:

```
barplot(VADeaths[, "Rural Male"],
        # ubah warna ouline menjadi steelblue
        border="steelblue",
        # ubah wana box
        col= c("grey", "yellow", "steelblue", "green", "orange"),
        # ubah nama grup dari A sampai E
        names.arg = LETTERS[1:5],
        # ubah orientasi menajadi horizontal
        horiz=TRUE)
```

Untuk bar plot dengan *multiple group*, tersedia dua pengaturan posisi yaitu *stacked bar plot*(menunjukkan proporsi penyusun pada masing-masing grup) dan *grouped bar plot*(melihat perbedaan individual pada masing-masing grup). Pada Gambar 3.6 dan Gambar 3.7 , disajikan kedua jenis bar plot tersebut.

```
# staked
barplot(VADeaths,
         col = c("lightblue", "mistyrose", "lightcyan",
                 "lavender", "cornsilk"),
        legend = rownames(VADeaths))
```

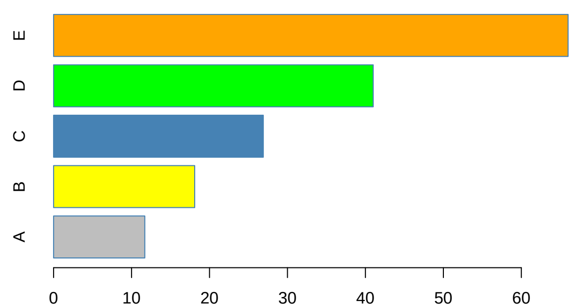

Figure 3.5: Kustomisasi bar plot

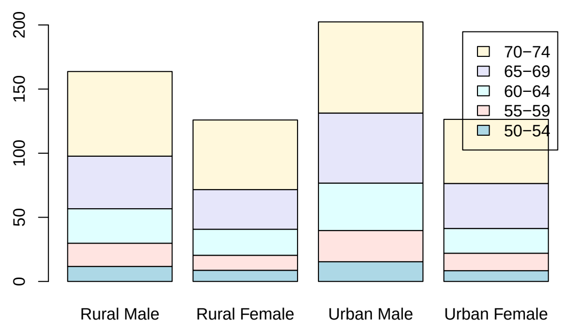

Figure 3.6: Stacked bar plot

```
# grouped
barplot(VADeaths,
         col = c("lightblue", "mistyrose", "lightcyan",
                 "lavender", "cornsilk"),
        legend = rownames(VADeaths), beside = TRUE)
```

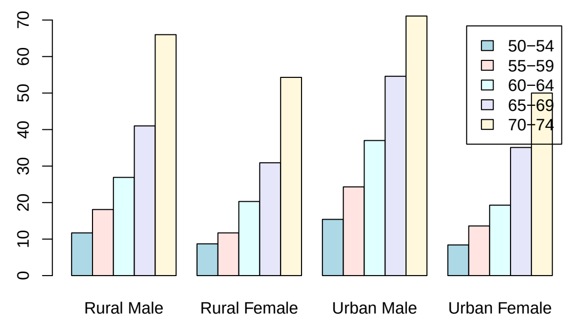

Figure 3.7: Grouped bar plot

3.2.2 Histogram dan Density Plot

Fungsi `hist()` dapat digunakan untuk membuat histogram pada `R`. Secara sederhana fungsi tersebut didefinisikan sebagai berikut:

```
hist(x, breaks="Sturges")
```

Catatan:

- **x**: vektor numerik
- **breaks**: *breakpoints* antar sel histogram.

Pada dataset `trees` akan dibuat histogram variabel `Height`. Untuk melakukannya jalankan sintaks berikut:

```
hist(trees$Height)
```

Output yang dihasilkan disajikan pada Gambar 3.8:

Figure 3.8: Histogram

Density plot pada `R` dapat dibuat menggunakan fungsi `density()`. Berbeda dengan fungsi `hist()`, fungsi ini tidak langsung menghasilkan grafik densitas. Fungsi `density()` hanya menghitung kernel densitas pada data. Densitas yang telah dihitung selanjutnya diplotkan menggunakan fungsi `plot()`. Berikut adalah sintaks dan output yang dihasilkan pada Gambar 3.9:

```
# menghitung kernel density
dens <- density(trees$Height)

# plot densitas dengan outline merah
plot(dens,col="red")
```

Kita juga dapat menambahkan grafik densitas pada histogram sehingga mempermudah pembacaan pada histogram. Untuk melakukannya kita perlu mengubah kernel histigram dari frekuensi menjadi density dengan menambahkan argumen `freq=FALSE` pada fungsi `hist()`. Selanjutnya tambahkan fungsi `polygon()` untuk memplotkan grafik densitas. Berikut adalah sintak dan output yang dihasilkan pada Gambar 3.10:

```
# menghitung kernel density
dens <- density(trees$Height)

# histogram
hist(trees$Height, freq=FALSE, col="steelblue")

# tambahkan density plot
polygon(dens, border="red")
```

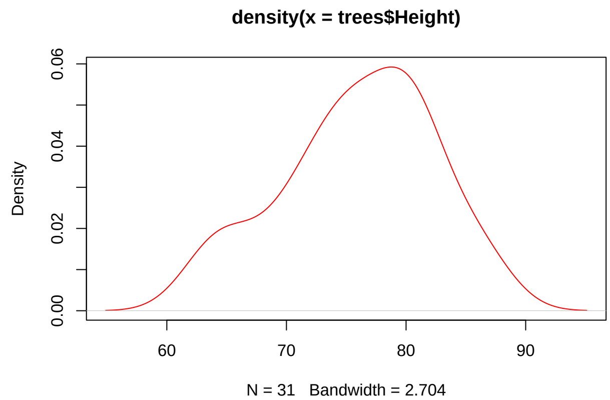

Figure 3.9: Density plot

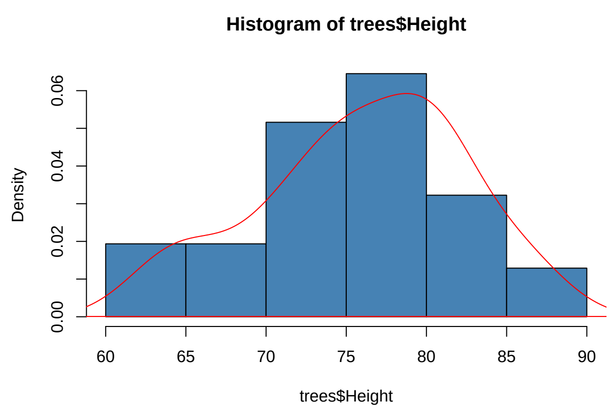

Figure 3.10: Density plot dan histogram

3.2.3 Box plot

Box plot pada `R` dapat dibuat menggunakan fungsi `boxplot()`. Berikut adalah sintaks untuk membuat boxplot variabel `Sepal.Lenght` pada dataset `iris` dan output yang dihasilkan pada Gambar 3.11:

```
boxplot(iris$Sepal.Length)
```

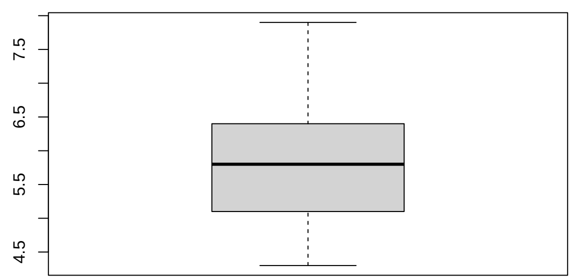

Figure 3.11: Boxplot variabel Sepal.Length

Boxplot juga dapat dibuat berdasarkan variabel factor. Hal ini berguna untuk melihat perbedaan ditribusi data pada masing-masing grup. Pada sintaks berikut dibuat boxplot berdasarkan variabel `Species`. Output yang dihasilkan disajikan pada Gambar 3.12:

```
boxplot(iris$Sepal.Length~iris$Species)
```

Kita juga dapat mengubah warna outline dan box pada boxplot. Berikut adalah contoh sintaks yang digunakan untuk melakukannya dan output yang dihasilkan disajikan pada Gambar 3.13:

```
boxplot(iris$Sepal.Length~iris$Species,
        # ubah warna outline menjadi steelblue
        border = "steelblue",
        # ubah warna box berdasarkan grup
        col= c("#999999", "#E69F00", "#56B4E9"))
```

Kita juga dapat membuat boxplot pada *multiple group*. Data yang digunakan untuk contoh tersebut adalah dataset `ToothGrowth`. Berikut adalah sintaks untuk memuat dataset tersebut:

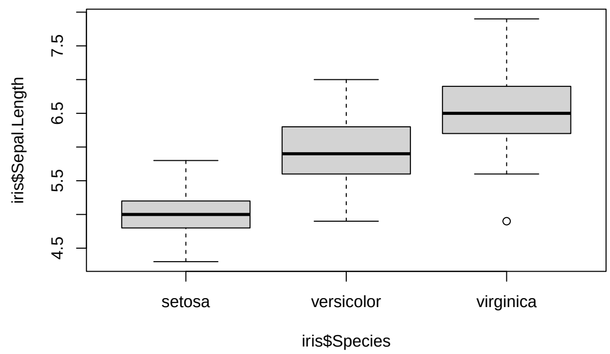

Figure 3.12: Boxplot berdasarkan variabel species

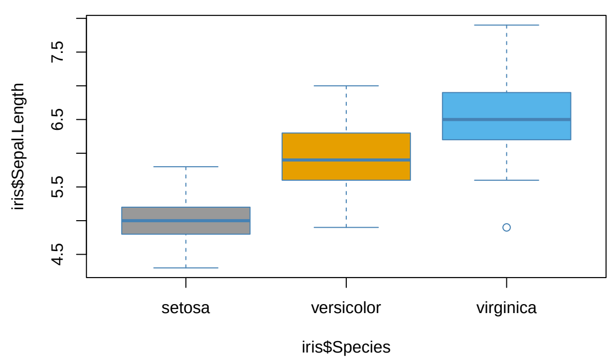

Figure 3.13: Boxplot dengan warna berdasarkan spesies

```r
# ubah variable dose menjadi factor
ToothGrowth$dose <- as.factor(ToothGrowth$dose)

# print
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Contoh sintaks dan output boxplot *multiple group* disajikan pada Gambar 3.14:

```r
boxplot(len ~ supp*dose, data = ToothGrowth,
        col = c("white", "steelblue"))
```

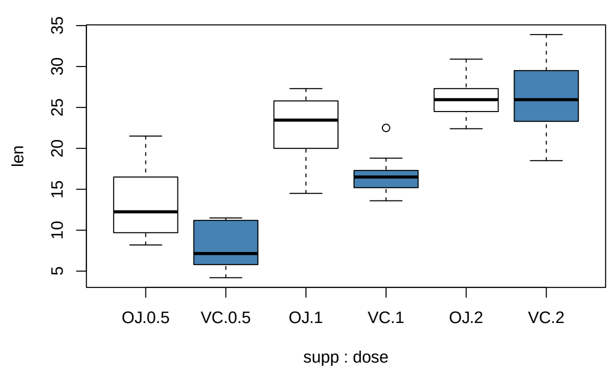

Figure 3.14: Boxplot multiple group

3.3 Kustomisasi Parameter Grafik

Pada bagian ini penulis akan menjelaskan cara untuk kustomisasi parameter grafik seperti:

a. menambahkan judul, legend, teks, axis, dan garis.
b. mengubah skala axis, simbol plot, jenis garis, dan warna.

3.3.1 Menambahkan Judul

Pada grafik di R, kita dapat menambahkan judul dengan dua cara, yaitu: pada plot melalui parameter dan melalui fungsi plot(). Kedua cara tersebut tidak berbeda satu sama lain pada parameter input.

Untuk menambahkan judul pada plot secara langsung, kita dapat menggunakan argumen tambahan sebagai berikut:

a. **main**: teks untuk judul.
b. **xlab**: teks untuk keterangan axis X.
c. **ylab**: teks untuk keterangan axis y.
d. **sub**: teks untuk sub-judul.

Berikut contoh sintaks penerapan masing-masing argumen tersebut beserta dengan output yang dihasilkan pada Gambar 3.15:

```
# menambahkan judul
barplot(c(2,5), main="Main title",
        xlab="X axis title",
        ylab="Y axis title",
        sub="Sub-title")
```

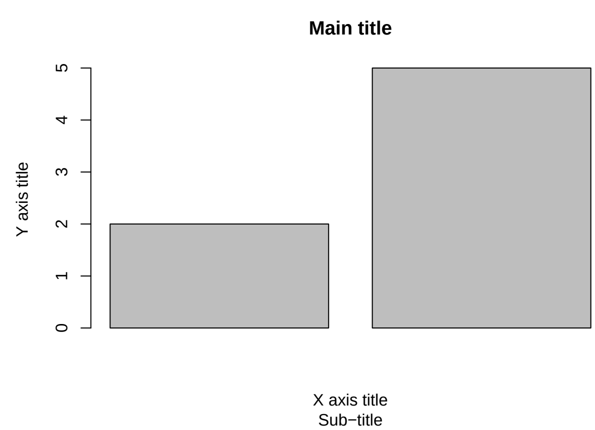

Figure 3.15: Menambahkan Judul

kita juga dapat melakukan kustomisasi pada warna, *font style*, dan ukuran font judul. Untuk melakukan kustomisasi pada warna pada judul, kita dapat menambahkan argumen sebagai berikut:

a. **col.main**: warna untuk judul.

b. **col.lab**: warna untuk keterangan axis.
c. **col.sub**: warna untuk sub-judul

Untuk kustomisasi font judul, kita dapat menambahkan argumen berikut:

a. **font.main**: *font style* untuk judul.
b. **font.lab**: *font style* untuk keterangan axis.
c. **font.sub**: *font style* untuk sub-judul.

Penting!!!

Nilai yang dapat dimasukkan antara lain:

- **1**: untuk teks normal.
- **2**: untuk teks cetak tebal.
- **3**: untuk teks cetak miring.
- **4**: untuk teks cetak tebal dan miring.
- **5**: untuk font simbol.

Sedangkan untuk ukuran font, kita dapat menambahkan variabel berikut:

a. **cex.main**: ukuran teks judul.
b. **cex.lab**: ukuran teks keterangan axis.
c. **cex.sub**: ukuran teks sub-judul.

Berikut sintaks penerapan seluruh argumen tersebut beserta output yang dihasilkan pada Gambar 3.16:

```
# menambahkan judul
barplot(c(2,5),
        # menambahkan judul
        main="Main title",
        xlab="X axis title",
        ylab="Y axis title",
        sub="Sub-title",
        # kustomisasi warna font
        col.main="red",
        col.lab="blue",
        col.sub="black",
        # kustomisasi font style
        font.main=4,
        font.lab=4,
        font.sub=4,
        # kustomisasi ukuran font
        cex.main=2,
        cex.lab=1.7,
        cex.sub=1.2)
```

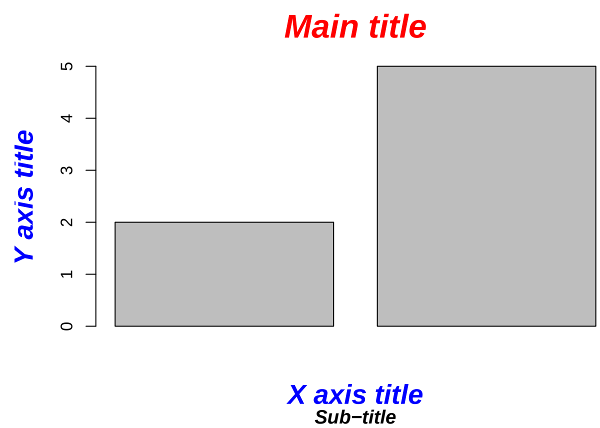

Figure 3.16: Menambahkan Judul (2)

Kita telah belajar bagaimana menambahkan judul langsung pada fungsi plot. Selain cara tersebut, telah penulis jelaskan bahwa kita dapat menambahkan judul melalui fungsi `title()`. argumen yang dimasukkan pada dasarnya tidak berbeda dengan ketika kita menambahkan judul secara langsung pada plot. Berikut adalah contoh sintaks dan output yang dihasilkan pada Gambar 3.17:

```
# menambahkan judul
barplot(c(2,5,8))

# menambahkan judul
title(main="Main title",
      xlab="X axis title",
      ylab="Y axis title",
      sub="Sub-title",
      # kustomisasi warna font
      col.main="red",
      col.lab="blue",
      col.sub="black",
      # kustomisasi font style
      font.main=4,
      font.lab=4,
      font.sub=4,
      # kustomisasi ukuran font
      cex.main=2,
      cex.lab=1.7,
      cex.sub=1.2)
```

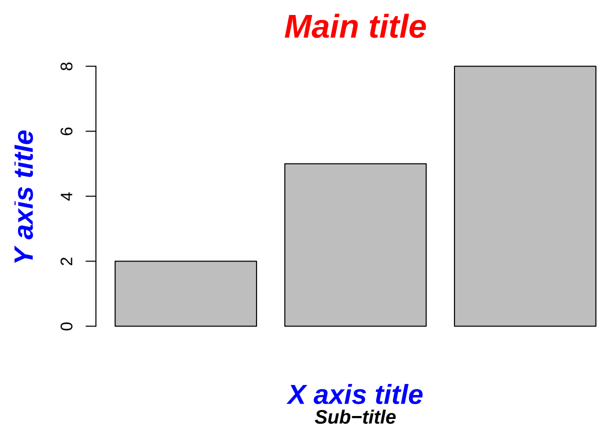

Figure 3.17: Menambahkan Judul (3)

3.3.2 Menambahkan Legend

Fungsi `legend()` pada `R` dapat digunakan untuk menambahkan legend pada grafik. Format sederhananya adalah sebagai berikut:

```
legend(x, y=NULL, legend, fill, col, bg)
```

Catatan:

- **x** dan **y**: koordinat yang digunakan untuk posisi legend.
- **legend**: teks pada legend
- **fill**: warna yang digunakan untuk mengisi box disamping teks legend.
- **col**: warna garis dan titik disamping teks legend.
- **bg**: warna latar belakang legend box.

Berikut adalah contoh sintaks dan ouput penerapan argumen disajikan pada Gambar 3.18:

```
# membuat vektor numerik
x <- c(1:10)
y <- x^2
z <- x*2

# membuat line plot
plot(x,y, type="o", col="red", lty=1)
```

```
# menambahkan line plot
lines(x,z, type="o", col="blue", lty=2)

# menambahkan legend
legend(1, 95, legend=c("Line 1", "Line 2"),
       col=c("red", "blue"), lty=1:2, cex=0.8)
```

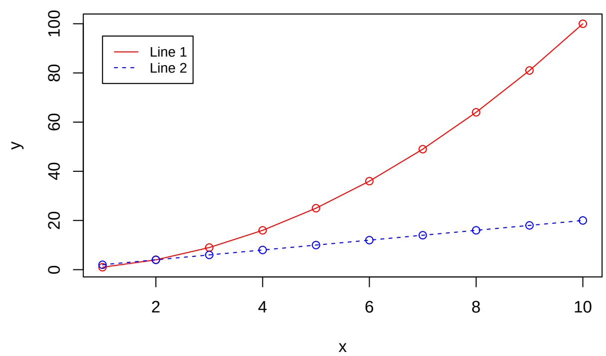

Figure 3.18: Menambahkan legend

Kita dapat menambahkan judul, merubah font, dan merubah warna backgroud pada legend. Argumen yang ditambahkan pada legend adalah sebagai berikut:

a. **title**: Judul legend
b. **text.font**: integer yang menunjukkan *font style* pada teks legend. Nilai yang dapat dimasukkan adalah sebagai berikut:
   - **1**: normal
   - **2**: cetak tebal
   - **3**: cetak miring
   - **4**: cetak tebal dan miring.
c. **bg**: warna background legend box.

Berikut adalah penerapan sintaks dan output yang dihasilkan pada Gambar 3.19:

```
# membuat line plot
plot(x,y, type="o", col="red", lty=1)

# menambahkan line plot
```

```
lines(x,z, type="o", col="blue", lty=2)

# menambahkan legend
legend(1, 95, legend=c("Line 1", "Line 2"),
       col=c("red", "blue"), lty=1:2, cex=0.8,
       title="Line types", text.font=4, bg='lightblue')
```

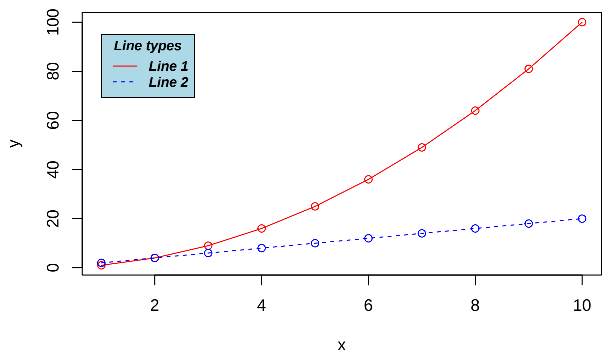

Figure 3.19: Menambahkan legend (2)

Kita dapat melakukan kustomisasi pada border dari legend melalui argumen `box.lty=`(jenis garis), `box.lwd=`(ukuran garis), dan `box.col=`(warna box). Berikut adalah penerapan argumen tersebut beserta output yang dihasilkan pada Gambar 3.20:

```
# membuat line plot
plot(x,y, type="o", col="red", lty=1)

# menambahkan line plot
lines(x,z, type="o", col="blue", lty=2)

# menambahkan legend
legend(1, 95, legend=c("Line 1", "Line 2"),
       col=c("red", "blue"), lty=1:2, cex=0.8,
       title="Line types", text.font=4, bg='white',
       box.lty=2, box.lwd=2, box.col="steelblue")
```

Selain menggunakan koordinat, kita juga dapat melakukan kustomisasi posisi legend menggunakan *keyword* seperti: bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". Sejumlah kustomisasi legend berdasarkan *keyword* disajikan pada Gambar 3.21:

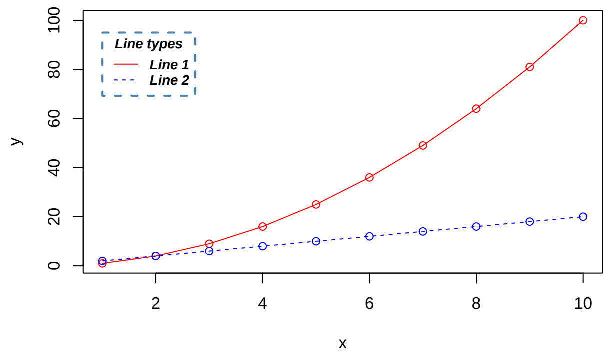

Figure 3.20: Menambahkan legend (3)

```
# plot
plot(x,y, type = "n")

# posisi kiri atas, inset =0.05
legend("topleft",
  legend = "(x,y)",
  title = "topleft, inset = .05",
  inset = 0.05)
# posisi atas
legend("top",
       legend = "(x,y)",
       title = "top")
# posisi kanan atas inset = .02
legend("topright",
       legend = "(x,y)",
       title = "topright, inset = .02",
       inset = 0.02)
# posisi kiri
legend("left",
       legend = "(x,y)",
       title = "left")
# posisi tengah
legend("center",
       legend = "(x,y)",
       title = "center")
# posisi kanan
legend("right",
```

```
       legend = "(x,y)",
       title = "right")
# posisi kiri bawah
legend("bottomleft",
       legend = "(x,y)",
       title = "bottomleft")
# posisi bawah
legend("bottom",
       legend = "(x,y)",
       title = "bottom")
# posisi kanan bawah
legend("bottomright",
       legend = "(x,y)",
       title = "bottomright")
```

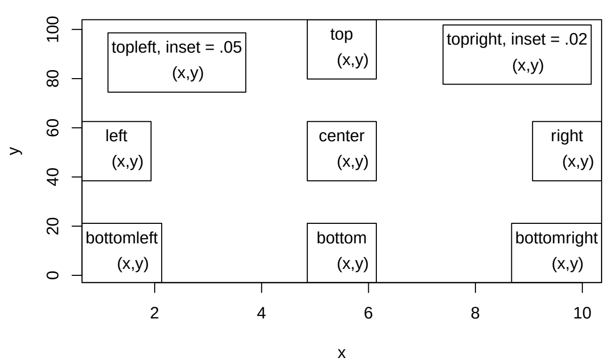

Figure 3.21: Kustomisasi posisi legend

3.3.3 Menambahkan Teks Pada Grafik

Teks pada grafik dapat kita tambahkan baik sebagai keterangan yang menunjukkan label suatu observasi, keterangan tambahan disekitar bingkai grafik, maupun sebuah persamaan yang ada pada bidang grafik. Untuk menambahkannya kita dapat menggunakan dua buah fungsi yaitu: `text()` dan `mtext()`.

FUngsi `text()` berguna untuk menambahkan teks di dalam bidang grafik seperti label titik observasi dan persamaan di dalam bidang grafik. Format yang digunakan adalah sebagai berikut:

```
text(x, y, labels)
```

Catatan:

- **x** dan **y**: vektor numerik yang menunjukkan koordinat posisi teks.
- **labels**: vektor karakter yang menunjukkan teks yang hendak ditulis.

Berikut adalah contoh sintaks untuk memberi label pada sejumlah data yang memiliki kriteria yang kita inginkan dan output yang dihasilkan pada Gambar 3.22:

```
# tandai observasi yang memiliki nilai
# mpg < 15 dan wt > 5
d <- mtcars[mtcars$wt >= 5 & mtcars$mpg <= 15, ]


# plot
plot(mtcars$wt, mtcars$mpg, main="Milage vs. Car Weight",
      xlab="Weight", ylab="Miles/(US) gallon")

# menambahkan text
text(d$wt, d$mpg,  row.names(d),
     cex=0.65, pos=3,col="red")
```

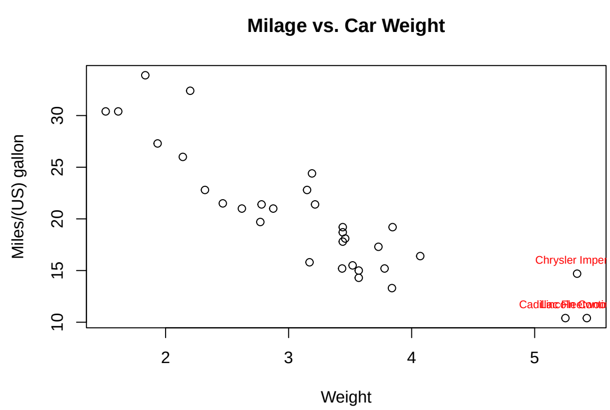

Figure 3.22: Menambahkan teks

Sedangkan sintaks berikut adalah contoh bagaimana menambahkan persamaan kedalam bidang grafik dan output yang dihasilkan pada Gambar 3.23:

```
plot(1:10, 1:10,
     main="text(...) examples\n~~~~~~~~~~~")
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(7, 4, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
```

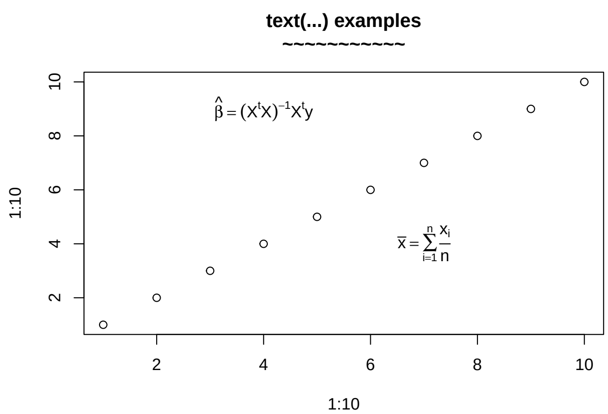

Figure 3.23: Menambahkan teks (2)

Fungsi `mtext()` berguna untuk menambahkan teks pada frame sekitar bidang grafik. Format yang digunakan adalah sebagai berikut:

```
mtext(text, side=3)
```

Catatan:

- **text**: teks yang akan ditulis.
- **side**: integer yang menunjukkan lokasi teks yang akan ditulis. Nilai yang dapat dimasukkan antara lain:
- 1: bawah
- 2: kiri
- 3: atas
- 4: kanan.

Berikut adalah contoh penerapan dan output yang dihasilkan pada Gambar 3.24:

```
plot(1:10, 1:10,
     main="mtext(...) examples\n~~~~~~~~~~~")
mtext("Magic function", side=3)
```

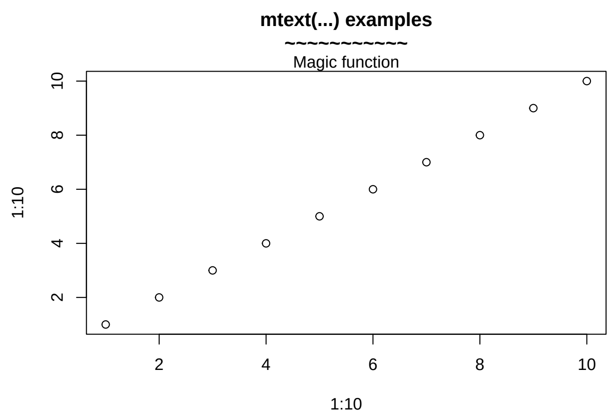

Figure 3.24: Menambahkan teks (3)

3.3.4 Menambahkan Garis Pada Plot

Fungsi `abline()` dapat digunakan untuk menamabahkan garis pada plot. Garis yang ditambahkan dapat berupa garis vertikal, horizontal, maupun garis regresi. Format yang digunakan adalah sebagi berikut:

```
abline(v=y)
```

Berikut adalah contoh sintaks bagaimana menambahkan garis pada sebuah plot dan output yang dihasilkan disajikan pada Gambar 3.25:

```
# membuat plot
plot(mtcars$wt, mtcars$mpg, main="Milage vs. Car Weight",
      xlab="Weight", ylab="Miles/(US) gallon")

# menambahkan garis vertikal di titik rata-rata weight
abline(v=mean(mtcars$wt), col="red", lwd=3, lty=2)

# menambahkan garis horizontal di titik rata-rata  mpg
abline(h=mean(mtcars$mpg), col="blue", lwd=3, lty=3)

# menambahkan garis regresi
abline(lm(mpg~wt, data=mtcars), lwd=4, lty=4)
```

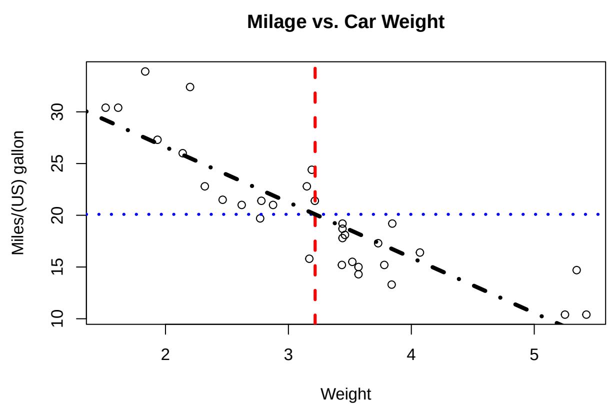

Figure 3.25: Menambahkan garis

3.3.5 Merubah Simbol plot dan Jenis Garis

Simbol plot (jenis titik) dapat diubah dengan menambahkan argumen `pch=` pada plot. Nilai yang dimasukkan pada argumen tersebut adalah integer dengan kemungkinan nilai sebagai berikut:

- pch = 0,square
- pch = 1,circle (default)
- pch = 2,triangle point up
- pch = 3,plus
- pch = 4,cross
- pch = 5,diamond
- pch = 6,triangle point down
- pch = 7,square cross
- pch = 8,star
- pch = 9,diamond plus
- pch = 10,circle plus
- pch = 11,triangles up and down
- pch = 12,square plus
- pch = 13,circle cross
- pch = 14,square and triangle down
- pch = 15, filled square
- pch = 16, filled circle
- pch = 17, filled triangle point-up
- pch = 18, filled diamond
- pch = 19, solid circle
- pch = 20,bullet (smaller circle)
- pch = 21, filled circle blue

- pch = 22, filled square blue
- pch = 23, filled diamond blue
- pch = 24, filled triangle point-up blue
- pch = 25, filled triangle point down blue

Untuk lebih memahami bentuk simbol tersebut, penulis akan menyajikan sintaks yang menampilkan seluruh simbol tersebut pada satu grafik. Output yang dihasilkan disajikan pada Gambar 3.26:

```
generateRPointShapes<-function(){
  # menentukan parameter plot
  oldPar<-par()
  par(font=2, mar=c(0.5,0,0,0))
  # produksi titik axis
  y=rev(c(rep(1,6),rep(2,5), rep(3,5), rep(4,5), rep(5,5)))
  x=c(rep(1:5,5),6)
  # plot seluruh titik dan label
  plot(x, y, pch = 0:25, cex=1.5, ylim=c(1,5.5), xlim=c(1,6.5),
       axes=FALSE, xlab="", ylab="", bg="blue")
  text(x, y, labels=0:25, pos=3)
  par(mar=oldPar$mar,font=oldPar$font )
}

# Print
generateRPointShapes()
```

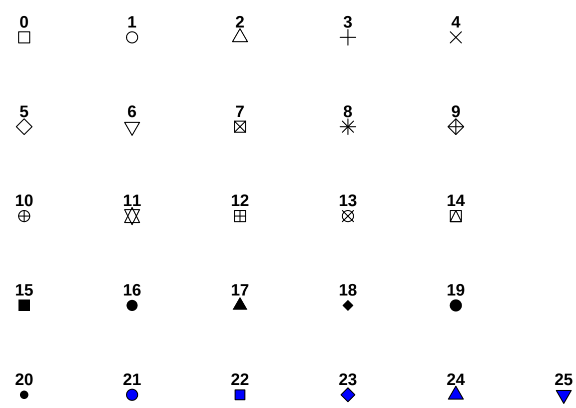

Figure 3.26: Symbol plot

Pada R kita juga dapat mengatur jenis garis yang akan ditampilkan pada plot dengan menambahkan argumen `lty=` (*line type*) pada fungsi plot. Nilai yang dapat dimasukkan adalah nilai integer. Keterangan masing-masing nilai tersebut adalah sebagai berikut:

- lty = 0, blank
- lty = 1, solid (default)
- lty = 2, dashed
- lty = 3, dotted
- lty = 4, dotdash
- lty = 5, longdash
- lty = 6, twodash

Untuk lebih memahaminya, pada sintaks berikut disajikan plot seluruh jenis garis tersebut beserta output yang dihasilkannya pada Gambar 3.27:

```
generateRLineTypes<-function(){
  oldPar<-par()
  par(font=2, mar=c(0,0,0,0))
  plot(1, pch="", ylim=c(0,6), xlim=c(0,0.7), axes = FALSE ,xlab="", ylab="")
  for(i in 0:6) lines(c(0.3,0.7), c(i,i), lty=i, lwd=3)
  text(rep(0.1,6), 0:6,
       labels=c("0.'blank'", "1.'solid'", "2.'dashed'", "3.'dotted'",
                "4.'dotdash'", "5.'longdash'", "6.'twodash'"))
  par(mar=oldPar$mar,font=oldPar$font )
}
generateRLineTypes()
```

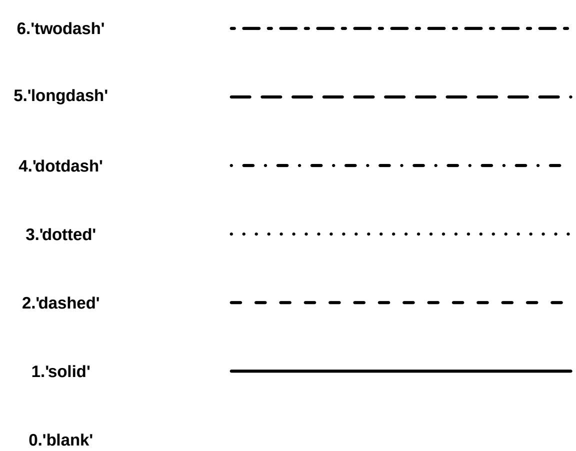

Figure 3.27: Line type

3.3.6 Mengatur Axis Plot

Kita dapat melakukan pengaturan lebih jauh terhadap axis, seperti: menambahkan axis tambahan pada atas dan bawah frame, mengubah rentang nilai axis, serta kustomisasi *tick mark* pada nilai axis. Hal ini diperlukan karena fungsi grafik dasar `R` tidak dapat mengatur axis secara otomatis saat plot baru ditambahkan pada plot pertama dan rentang nilai plot baru lebih besar dibanding plot pertama, sehingga sebagian nilai plot baru tidak ditampilkan pada hasil akhir.

Untuk menambahkan axis pada `R` kita dapat menambahkan fungsi `axis()` setelah plot dilakukan. Format yang digunakan adalah sebagai berikut:

```
axis(side, at=NULL, labels=TRUE)
```

Catatan:

- **side**: nilai integer yang mengidikasikan posisi axix yang hendak ditambahkan. Nilai yang dapat dimasukkan adalah sebagai berikut:
  - 1: bawah
  - 2: kiri
  - 3: atas
  - 4: kanan.
- **at**: titik dimana *tick-mark* hendak digambarkan. Nilai yang dapat dimasukkan sama dengan `side`.
- **labels**: Teks label *tick-mark*. Dapat juga secara logis menentukan apakah anotasi harus dibuat pada *tick mark*.

Berikut contoh sintaks penerapan fungsi tersebut dan output yang dihasilkan pada Gambar 3.28:

```
# membuat vektor numerik
x <- c(1:4)
y <- x^2

# plot
plot(x, y, pch=18, col="red", type="b",
     frame=FALSE, xaxt="n") # Remove x axis

# menambahkan axis
# bawah
axis(1, 1:4, LETTERS[1:4], col.axis="blue")
# atas
axis(3, col = "darkgreen", lty = 2, lwd = 0.5)
```

```
# kanan
axis(4, col = "violet", col.axis = "dark violet", lwd = 2)
```

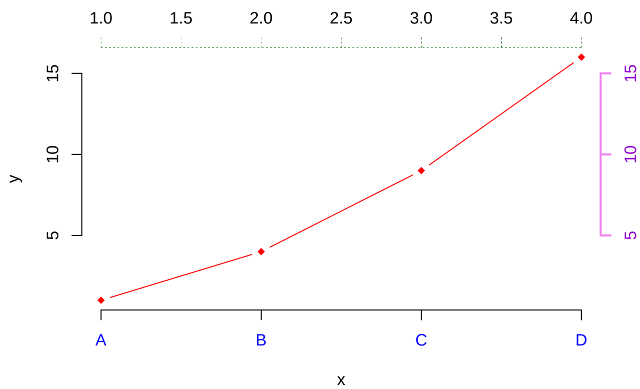

Figure 3.28: Modifikasi axis

Kita dapat mengubah rentang nilai pada axis menggunakan fungsi `xlim()` dan `ylim()` yang menyatakan vektor nilai masimum dan minimum rentang. Selain itu kita dapat juga melakukan tranformasi baik pada sumbu x dan sumbu y. Berikut adalah argumen yang dapat ditambahkan pada fungsi grafik:

- **xlim**: limit nilai sumbu x dengan format: `xlim(min, max)`.
- **ylim**: limit nilai sumbu x dengan format: `ylim(min, max)`.

Untuk transformasi skala log, kita dapat menambahkan argumen berikut:

- **log="x"**: transformasi log sumbu x.
- **log="y"**: transformasi log sumbu y.
- **log="xy"**: transformasi log sumbu x dan y.

Berikut adalah contoh sintaks penerapan argumen tersebut beserta output yang dihasilkan pada Gambar 3.29:

```
# membagi jendela grafik menjadi 1 baris dan 3 kolom
par(mfrow=c(1,3))

# membuat vektor numerik
x<-c(1:10); y<-x*x
```

```
# simple plot
plot(x, y)

# plot dengan pengaturan rentang skala
plot(x, y, xlim=c(1,15), ylim=c(1,150))

# plot dengan transformasi skala log
plot(x, y, log="y")
```

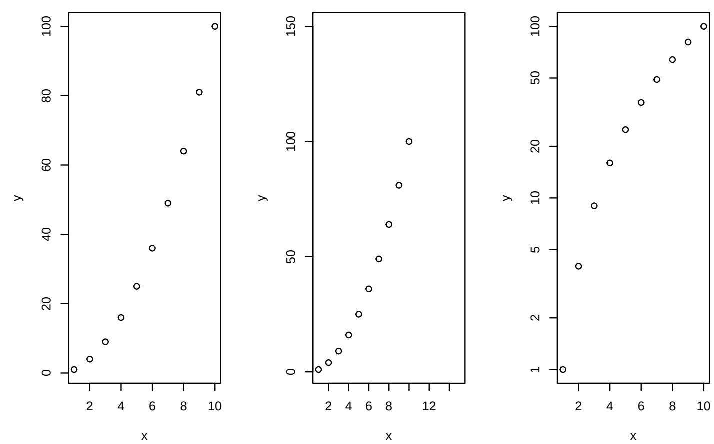

Figure 3.29: Mengubah rentang dan skala axis

Kita dapat melakukan kustomisasi pada *tick mark*. Kustomisasi yang dapat dilakukan adalah merubah warna, *font style*, ukuran font, orientasi, serta menyembunyikan *tick mark*.

Argumen yang ditambahkan adalah sebagai berikut:

- **col.axis**: warna *tick mark*.
- **font.axis**: integer yang menunjukkan *font style*. Sama dengan pengaturan judul.
- **cex.axis**: pengaturan ukuran *tick mark*.
- **las**: mengatur orientasi *tick mark*. Nilai yang dapat dimasukkan adalah sebagai berikut:
  - **0**: paralel terhadap posisi axis (default)
  - **1**: selalu horizontal

  - **2**: selalu perpendikular dengan posisi axis
  - **3**: selalu vertikal

- **xaxt** dan **yaxt**: karakter untuk menunjukkan apakah axis akan ditampilkan atau tidak. nilai dapat berupa "n"(sembunyika) dan "s"(tampilkan).

Berikut adalah contoh penerapan argumen tersebut beserta output pada Gambar 3.30:

```
# membuat vektor numerik
x<-c(1:10); y<-x*x

# plot
plot(x,y,
     # warna
     col.axis="red",
     # font style
     font.axis=2,
     # ukuran
     cex=1.5,
     # orientasi
     las=3,
     # sembunyikan sumbu x
     xaxt="n")
```

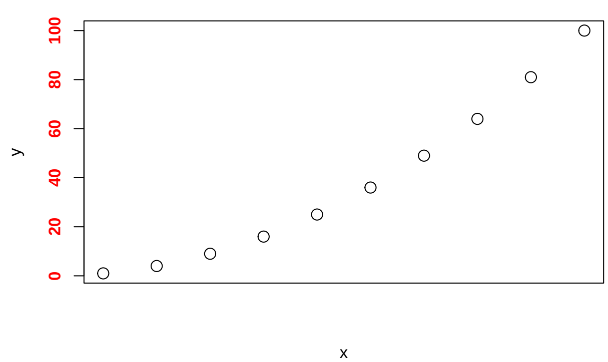

Figure 3.30: Kustomisasi tick mark

3.3.7 Mengatur Warna

Pada fungsi dasar `R`, warna dapat diatur dengan mengetikkan nama warna maupun kode hexadesimal. Selain itu kita juga dapat menamambahkan warna lain melalui library lain yang tidak dijelaskan pada chapter ini.

Untuk penggunaan warna hexadesima kita perlu mengetikkan "#" yang diukuti oleh 6 kode warna. Untuk memperlajari kode-kode dan warna yang dihasilkan, silahkan pembaca mengunjungi situs http://www.visibone.com/.

Pada sintaks berikut disajikan visualisasi nama-nama warna bawaan yang ada pada `R`. Output yang dihasilkan disajikan pada Gambar 3.31:

```
showCols <- function(cl=colors(), bg = "grey",
                     cex = 0.75, rot = 30) {
    m <- ceiling(sqrt(n <-length(cl)))
    length(cl) <- m*m; cm <- matrix(cl, m)
    require("grid")
    grid.newpage(); vp <- viewport(w = .92, h = .92)
    grid.rect(gp=gpar(fill=bg))
    grid.text(cm, x = col(cm)/m, y = rev(row(cm))/m, rot = rot,
              vp=vp, gp=gpar(cex = cex, col = cm))
}

# print 60 nama warna pertama
showCols(bg="gray20", cl=colors()[1:60], rot=30, cex=0.9)
```

```
## Loading required package: grid
```

Figure 3.31: Nama warna

3.4 Plot Dua dan Tiga Dimensi

`R` dapat digunakan untuk memproduksi visualisasi pada skala 2 dan 3 dimensi. Untuk proyeksi 2 dimensi, fungsi yang digunakan adalah `image()` atau `contour()`. Untuk informasi lebih lanjut terkait fungsi tersebut pembaca dapat mengakses menu bantuan. Pada sintak berikut diberikan contoh bagaimana cara memproduksi visualisasi dua dimensi menggunakan kedua fungsi tersebut:

```
n <- 1:20
x <- sin(n)
y <- cos(n)*exp(-n/3)
z <- outer(x,y)
par(mar=c(3,3,1.5,1.5), mex=0.8, mgp=c(2,0.5,0), tcl=0.3)
par(mfrow=c(1,2))

# plot pertama
image(z, col=gray(1:10/10))

# plot kedua
contour(z)
```

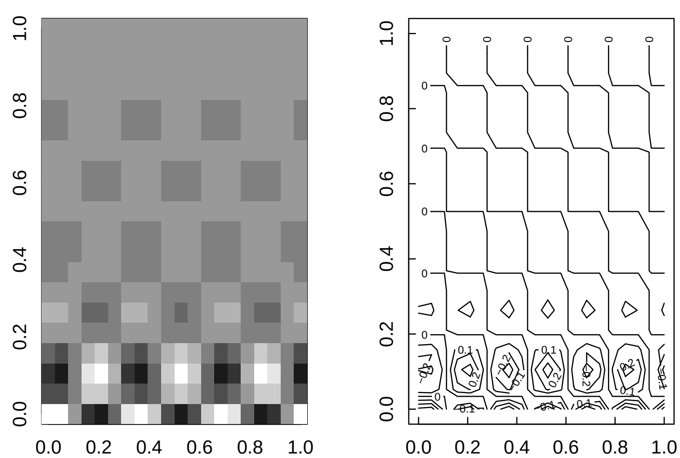

Figure 3.32: image plot (kiri) dan contour plot (kanan)

```
par(mfrow=c(1,1))
```

Proyeksi 3 dimensi dapat dilakukan menggunakan fungsi `persp()`. Sudut penglihatan dapat diatur melalui argumen`theta` (sudut) dan `phi()` (rotasi).

Sintaks berikut merupakan contoh bagaimana cara menghasilkan visualisasi 3 dimensi dari data yang telah diproduksi sebelumnya:

```
par(mar=c(3,3,1.5,1.5), mex=0.8, mgp=c(2,0.5,0), tcl=0.3)
par(mfrow=c(1,2))

# plot pertama
persp(n,n,z, theta=45, phi=20)

# plot kedua
persp(n,n,z, theta=45, phi=20, shade=0.5)
```

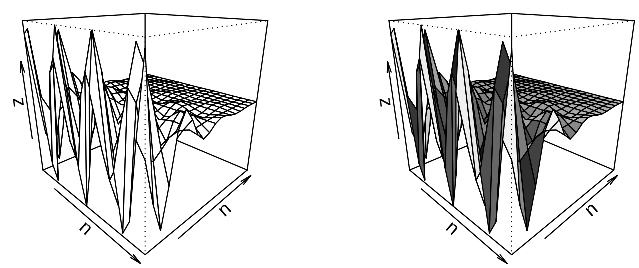

Figure 3.33: proyeksi 3 dimensi (kanan) dan proyeksi 3 dimensi dengan pewarnaan

```
par(mfrow=c(1,1))
```

3.5 Referensi

1. Maindonald, J.H. 2008. **Using R for Data Analysis and Graphics Introduction, Code and Commentary**. Centre for Mathematics and Its Applications Australian National University.
2. Scherber, C. 2007. **An introduction to statistical data analysis using R**. R_Manual Goettingen.
3. STHDA. **R Base Graphs**. http://www.sthda.com/english/wiki/r-base-graphs

4. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.

Chapter 4

Pemrograman dan Fungsi

Kita telah membahas dasar-dasar kalkulasi menggunakan `R` pada Chapter 2. Pada Chapter 4 kita akan membahas dasar pemrograman menggunakan `R`. Pada chapter ini kita juga akan membahas bagaimana kita dapat membentuk suatu fungsi menggunakan `R` untuk pekerjaan yang berulang-ulang.

4.1 Loop

Loop merupakan kode program yang berulang-ulang. *Loop* berguna saat kita ingin melakukan sebuah perintah yang perlu dijalankan berulang-ulang seperti melakukan perhitungan maupaun melakukan visualisasi terhadap banyak variabel secara serentak. Hal ini tentu saja membantu kita karena kita tidak perlu menulis sejumlah sintaks yang berulang-ulang. Kita hanya perlu mengatur *statement* berdasarkan hasil yang kita harapkan.

Pada `R` bentuk *loop* dapat bermacam-macam ("*for loop*","*while loop*", dll). `R` menyederhanakan bentuk *loop* ini dengan menyediakan sejumlah fungsi seperti `apply()`,`tapply()`, dll. Sehingga `loop` jarang sekali muncul dalam kode `R`. Sehingga `R` sering disebut sebagai *loopless loop*.

Meski *loop* jarang muncul bukan berarti kita tidak akan melakukannya. Terkadang saat kita melakukan komputasi statistik atau matematik dan belum terdapat *library* yang mendukung proses tersebut, sering kali kita akan membuat sintaks sendiri berdasarkan algoritma metode tersebut. Pada algoritma tersebut sering pula terdapat *loop* yang diperlukan selama proses perhitungan. Secara sederhana diagram umum loop ditampilkan pada Gambar 4.1

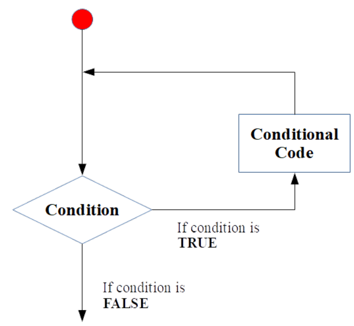

Figure 4.1: Diagram umum loop (sumber: Primartha, 2018).

4.1.1 For Loop

Mengulangi sebuah *statement* atau sekelompok *statement* sebanyak nilai yang ditentukan di awal. Jadi operasi akan terus dilakukan sampai dengan jumlah yang telah ditetapkan di awal atau dengan kata lain tes kondisi (Jika jumlah pengulangan telah cukup) hanya akan dilakukan di akhir. Secara sederhana bentuk dari *for loop* dapat dituliskan sebagai berikut:

```
for (value in vector){
  statements
}
```

Berikut adalah contoh sintaks penerapan *for loop*:

```
# Membuat vektor numerik
vektor <- c(1:5)

# loop
for(i in vektor){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Loop akan dimulai dari blok *statement* *for* sampai dengan `print(i)`. Berdasarkan *loop* pada contoh tersebut, *loop* hanya dilakukan sebanyak 5 kali sesuai dengan jumlah vektor yang ada.

4.1.2 While Loop

While loop merupakan loop yang digunakan ketika kita telah menetapkan *stop condition* sebelumnya. Blok *statement*/kode yang sama akan terus dijalankan sampai *stop condition* ini tercapai. *Stop condition* akan di cek sebelum melakukan proses *loop*. Berikut adalah pola dari *while loop* dapat dituliskan sebagai berikut:

```
while (test_expression){
  statement
}
```

Berikut adalah contoh penerapan dari *while loop*:

```
coba <- c("Contoh")
counter <- 1

# loop
while (counter<5){
  # print vektor
  print(coba)
  # tambahkan nilai counter sehingga proses terus berlangsung sampai counter = 5
  counter <- counter + 1
}
```

```
## [1] "Contoh"
## [1] "Contoh"
## [1] "Contoh"
## [1] "Contoh"
```

Loop akan dimulai dari blok *statement while* sampai dengan *counter* <- 1. *Loop* hanya akan dilakukan sepanjang nilai *counter* < 5.

4.1.3 Repeat Loop

Repeat loop akan menjalankan *statement*/kode yang sama berulang-ulang hingga *stop condition* tercapai. Berikut adalah pola dari *repeat loop*.

```
repeat {
  commands
  if(condition){
    break
  }
}
```

Berikut adalah contoh penerapan dari *repeat loop*:

```
coba <- c("contoh")
counter <- 1
repeat {
  print(coba)
  counter <- counter + 1
  if(counter < 5){
break
  }
}
```

```
## [1] "contoh"
```

Loop akan dimulai dari blok *statement while* sampai dengan *break*. *Loop* hanya akan dilakukan sepanjang nilai *counter* < 5. Hasil yang diperoleh berbeda dengan *while loop*, dimana kita memperoleh 4 buah kata "contoh". Hal ini disebabkan karena *repeat loop* melakukan pengecekan *stop condition* tidak di awal loop seperti *while loop* sehingga berapapun nilainya, selama nilainya sesuai dengan *stop condition* maka *loop* akan dihentikan. Hal ini berbeda dengan *while loop* dimana proses dilakukan berulang-ulang sampai jumlahnya mendekati *stop condition*.

4.1.4 Break

Break sebenarnya bukan bagian dari *loop*, namun sering digunakan dalam *loop*. *Break* dapat digunakan pada *loop* manakala dirasa perlu, yaitu saat kondisi yang disyaratkan pada *break* tercapai.

Berikut adalah contoh penerapan *break* pada beberapa jenis *loop*.

```
# for loop
a = c(2,4,6,8,10,12,14)
for(i in a){
  if(i>8){
    break
  }
  print(i)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
```

```
# while loop
a = 2
b = 4
while(a<7){
  print(a)
  a = a +1
  if(b+a>10){
    break
  }
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

```
# repeat loop
a = 1
repeat{
  print(a)
  a = a+1
  if(a>6){
    break
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

4.2 Loop Menggunakan Apply Family Function

Penggunaan loop sangat membantu kita dalam melakukan proses perhitungan berulang. Namun, metode ini tidak cukup ringkas dalam penerapannya dan perlu penulisan sintaks yang cukup panjang untuk menyelesaikan sebuah kasus yang kita inginkan. Berikut adalah sebuah sintaks yang digunakan untuk menghitung nilai mean pada suatu dataset:

```
# subset data iris
sub_iris <- iris[,-5]
# membuat vektor untuk menyimpan hasil loop
a <- rep(NA,4)
# loop
for(i in 1:length(sub_iris)){
  a[i]<-mean(sub_iris[,i])
}
# print
a
```

```
## [1] 5.843 3.057 3.758 1.199
```

```
class(a) # cek kelas objek
```

```
## [1] "numeric"
```

Metode alternatif lain untuk melakukan loop suatu fungsi adalah dengan menggunakan Apply function family. Metode ini memungkinkan kita untuk melakukan loop suatu fungsi tanpa perlu menuliskan sintaks loop. Berikut adalah beberapa fungsi dari apply family yang nantinya akan sering kita gunakan:

- `apply()`: fungsi generik yang mengaplikasikan fungsi kepada kolom atau baris pada matriks atau secara lebih general aplikasi dilakukan pada dimensi untuk jenis data array.
- `lapply()`: fungsi apply yang bekerja pada jenis data list dan memberikan output berupa list juga.
- `sapply()`: bentuk sederhana dari lapply yang menghasilkan output berupa matriks atau vektor.
- `vapply()`: disebut juga *verified apply* (memungkinkan untuk menghasilkan output dengan jenis data yang telah ditentukan sebelumnya).
- `tapply()`: *tagged apply* dimana dimana tag menentukan subset dari data.

4.2.1 Apply

Fungsi `apply()` bekerja dengan jenis data matrik atau array (jenis data homogen). Kita dapat melakukan spesifikasi apakah suatu fungsi hanya akan bekerja pada kolom saja, baris saja atau keduanya. Format fungsi ini adalah sebagai berikut:

```
apply(X, MARGIN, FUN, ...)
```

> **Catatan:**
>
> - **X**: matriks atau array
> - **MARGIN**: menentukan bagaimana fungsi bekerja terhadap matriks atau array. Jika nilai yang diinputkan 1, maka fungsi akan bekerja pada masing-masing baris pada matriks. Jika nilainya 2, maka fungsi akan bekerja pada tiap kolom pada matriks.
> - **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function.
> - **...**: opsional argumen pada fungsi yang digunakan.

Berikut adalah contoh bagaimana aplikasi fungsi tersebut pada matriks:

```
## membuat matriks
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
x # print
```

```
##      x1 x2
## [1,]  3  4
## [2,]  3  3
## [3,]  3  2
## [4,]  3  1
## [5,]  3  2
## [6,]  3  3
## [7,]  3  4
## [8,]  3  5
```

```
class(x) # cek kelas objek
```

```
## [1] "matrix" "array"
```

```
## menghitung mean masing-masing kolom
apply(x, MARGIN=2 ,FUN=mean, trim=0.2, na.rm=TRUE)
```

```
## x1 x2
##  3  3
```

```r
## menghitung range pada masing-masing baris
## menggunakan user define function
apply(x, MARGIN=1,
      FUN=function(x){
        max(x)-min(x)
      })
```

```
## [1] 1 0 1 2 1 0 1 2
```

4.2.2 lapply

Fungsi ini melakukan loop fungsi terhadap input data berupa list. Output yang dihasilkan juga merupakan list dengan panjang list yang sama dengan yang diinputkan. Format yang digunakan adalah sebagai berikut:

```r
lapply(X, FUN, ...)
```

> **Catatan:**
>
> - **X**: vektor, data frame atau list
> - **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
> - **...**: opsional argumen pada fungsi yang digunakan.

Berikut adalah contoh penerapan fungsi lapply:

```r
## Membuat list
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
x # print
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $beta
## [1]  0.04979  0.13534  0.36788  1.00000  2.71828
## [6]  7.38906 20.08554
##
## $logic
## [1]  TRUE FALSE FALSE  TRUE
```

```r
class(x) # cek kelas objek
```

```
## [1] "list"
```

```r
## Menghitung nilai mean pada masing-masing baris lits
lapply(x, FUN=mean)
```

```
## $a
## [1] 5.5
##
## $beta
## [1] 4.535
##
## $logic
## [1] 0.5
```

```r
## Menghitung mean tiap kolom dataset iris
lapply(iris, FUN=mean)
```

```
## Warning in mean.default(X[[i]], ...): argument is not
## numeric or logical: returning NA
```

```
## $Sepal.Length
## [1] 5.843
##
## $Sepal.Width
## [1] 3.057
##
## $Petal.Length
## [1] 3.758
##
## $Petal.Width
## [1] 1.199
##
## $Species
## [1] NA
```

```r
## Mengalikan elemen vektor dengan suatu nilai
y <- c(1:5)
lapply(y, FUN=function(x){x*5})
```

```
## [[1]]
```

```
## [1] 5
##
## [[2]]
## [1] 10
##
## [[3]]
## [1] 15
##
## [[4]]
## [1] 20
##
## [[5]]
## [1] 25
```

```r
## Mengubah output menjadi vektor
unlist(lapply(y, FUN=function(x){x*5}))
```

```
## [1]  5 10 15 20 25
```

4.2.3 sapply

Fungsi `sapply()` merupakan bentuk lain dari fungsi `lapply()`. Perbedaanya terletak pada output default yang dihasilkan. Secara default `sapply()` menerima input utama berupa list (dapat pula dataframe atau vektor), namun tidak seperti `lapply()` jenis data output yang dihasilkan adalah vektor. Untuk mengubah output menjadi list perlu argumen tambahan berupa `simplify=FALSE`. Format fungsi tersebut adalah sebagai berikut:

```r
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

> **Catatan:**
>
> - **X**: vektor, data frame atau list
> - **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
> - **...**: opsional argumen pada fungsi yang digunakan.
> - **simplify**: logical. Jika nilainya `TRUE` maka output yang dihasilkan adalah bentuk sederhana dari vektor, matrix atau array.
> - **USE.NAMES**: jika list memiliki nama pada setiap elemennya, maka nama elemen tersebut akan secara default ditampilkan.

Berikut adalah contoh penerapannya:

```
## membuat list
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))

## menghitung nilai mean setiap elemen
sapply(x, FUN=mean)
```

```
##     a  beta logic
## 5.500 4.535 0.500
```

```
## menghitung nilai mean dengan output list
sapply(x, FUN=mean, simplify=FALSE)
```

```
## $a
## [1] 5.5
##
## $beta
## [1] 4.535
##
## $logic
## [1] 0.5
```

```
## summary objek dataframe
sapply(mtcars, FUN=summary)
```

```
##           mpg   cyl  disp    hp  drat    wt  qsec
## Min.    10.40 4.000  71.1  52.0 2.760 1.513 14.50
## 1st Qu. 15.43 4.000 120.8  96.5 3.080 2.581 16.89
## Median  19.20 6.000 196.3 123.0 3.695 3.325 17.71
## Mean    20.09 6.188 230.7 146.7 3.597 3.217 17.85
## 3rd Qu. 22.80 8.000 326.0 180.0 3.920 3.610 18.90
## Max.    33.90 8.000 472.0 335.0 4.930 5.424 22.90
##             vs     am  gear  carb
## Min.    0.0000 0.0000 3.000 1.000
## 1st Qu. 0.0000 0.0000 3.000 2.000
## Median  0.0000 0.0000 4.000 2.000
## Mean    0.4375 0.4062 3.688 2.812
## 3rd Qu. 1.0000 1.0000 4.000 4.000
## Max.    1.0000 1.0000 5.000 8.000
```

```
## summary objek list
a <- list(mobil=mtcars, anggrek=iris)
sapply(a, FUN=summary)
```

```
## $mobil
##       mpg             cyl             disp
##  Min.   :10.4   Min.   :4.00   Min.   : 71.1
##  1st Qu.:15.4   1st Qu.:4.00   1st Qu.:120.8
##  Median :19.2   Median :6.00   Median :196.3
##  Mean   :20.1   Mean   :6.19   Mean   :230.7
##  3rd Qu.:22.8   3rd Qu.:8.00   3rd Qu.:326.0
##  Max.   :33.9   Max.   :8.00   Max.   :472.0
##        hp             drat             wt
##  Min.   : 52.0   Min.   :2.76   Min.   :1.51
##  1st Qu.: 96.5   1st Qu.:3.08   1st Qu.:2.58
##  Median :123.0   Median :3.69   Median :3.33
##  Mean   :146.7   Mean   :3.60   Mean   :3.22
##  3rd Qu.:180.0   3rd Qu.:3.92   3rd Qu.:3.61
##  Max.   :335.0   Max.   :4.93   Max.   :5.42
##       qsec             vs               am
##  Min.   :14.5   Min.   :0.000   Min.   :0.000
##  1st Qu.:16.9   1st Qu.:0.000   1st Qu.:0.000
##  Median :17.7   Median :0.000   Median :0.000
##  Mean   :17.8   Mean   :0.438   Mean   :0.406
##  3rd Qu.:18.9   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :22.9   Max.   :1.000   Max.   :1.000
##       gear           carb
##  Min.   :3.00   Min.   :1.00
##  1st Qu.:3.00   1st Qu.:2.00
##  Median :4.00   Median :2.00
##  Mean   :3.69   Mean   :2.81
##  3rd Qu.:4.00   3rd Qu.:4.00
##  Max.   :5.00   Max.   :8.00
##
## $anggrek
##   Sepal.Length    Sepal.Width     Petal.Length
##  Min.   :4.30   Min.   :2.00   Min.   :1.00
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60
##  Median :5.80   Median :3.00   Median :4.35
##  Mean   :5.84   Mean   :3.06   Mean   :3.76
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10
##  Max.   :7.90   Max.   :4.40   Max.   :6.90
##   Petal.Width          Species
##  Min.   :0.1   setosa    :50
##  1st Qu.:0.3   versicolor:50
##  Median :1.3   virginica :50
##  Mean   :1.2
##  3rd Qu.:1.8
##  Max.   :2.5
```

4.2.4 vapply

Funsgi ini merupakan bentuk lain dari `sapply()`. Bedanya secara kecepatan proses fungsi ini lebih cepat dari `sapply()`. Hal yang menarik dari fungsi ini kita dapat menambahkan argumen `FUN.VALUE`. pada argumen ini kita memasukkan vektor berupa output fungsi yang diinginkan. Perbedaan lainnya adalah output yang dihasilkan hanya berupa matriks atau array. Format dari fungsi ini adalah sebagai berikut:

```
vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)
```

Catatan:

- **X**: vektor, data frame atau list
- **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
- **FUN.VALUE**: vektor, template dari return value FUN.
- **...**: opsional argumen pada fungsi yang digunakan.
- **USE.NAMES**: jika list memiliki nama pada setiap elemennya, maka nama elemen tersebut akan secara default ditampilkan.

Berikut adalah contoh penerapannya:

```
## membuat list
x <- sapply(3:9, seq)
x # print
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 1 2 3 4
##
## [[3]]
## [1] 1 2 3 4 5
##
## [[4]]
## [1] 1 2 3 4 5 6
##
## [[5]]
## [1] 1 2 3 4 5 6 7
##
```

```
## [[6]]
## [1] 1 2 3 4 5 6 7 8
##
## [[7]]
## [1] 1 2 3 4 5 6 7 8 9
```

```
## membuat ringkasan data pada tiap elemen list
vapply(x, fivenum,
       c(Min. = 0, "1st Qu." = 0,
         Median = 0, "3rd Qu." = 0, Max. = 0))
```

```
##         [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## Min.     1.0  1.0    1  1.0  1.0  1.0    1
## 1st Qu.  1.5  1.5    2  2.0  2.5  2.5    3
## Median   2.0  2.5    3  3.5  4.0  4.5    5
## 3rd Qu.  2.5  3.5    4  5.0  5.5  6.5    7
## Max.     3.0  4.0    5  6.0  7.0  8.0    9
```

```
## membuat ringkasan data pada tiap kolom dataframe
vapply(mtcars, summary,
       c(Min. = 0, "1st Qu." = 0,
         Median = 0, "3rd Qu." = 0, Max. = 0, Mean=0))
```

```
##           mpg   cyl  disp    hp  drat    wt  qsec
## Min.    10.40 4.000  71.1  52.0 2.760 1.513 14.50
## 1st Qu. 15.43 4.000 120.8  96.5 3.080 2.581 16.89
## Median  19.20 6.000 196.3 123.0 3.695 3.325 17.71
## 3rd Qu. 20.09 6.188 230.7 146.7 3.597 3.217 17.85
## Max.    22.80 8.000 326.0 180.0 3.920 3.610 18.90
## Mean    33.90 8.000 472.0 335.0 4.930 5.424 22.90
##             vs     am  gear  carb
## Min.    0.0000 0.0000 3.000 1.000
## 1st Qu. 0.0000 0.0000 3.000 2.000
## Median  0.0000 0.0000 4.000 2.000
## 3rd Qu. 0.4375 0.4062 3.688 2.812
## Max.    1.0000 1.0000 4.000 4.000
## Mean    1.0000 1.0000 5.000 8.000
```

4.2.5 tapply

Fungsi ini sangat berguna jika pembaca ingin menghitung suatu nilai misalnya mean berdasarkan grup data atau factor. Format fungsi ini adalah sebagi berikut:

```
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

> **Catatan:**
>
> - **X**: vektor, data frame atau list
> - **INDEX**: list satu atau beberapa factor yang memiliki panjang sama dengan X.
> - **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
> - **...**: opsional argumen pada fungsi yang digunakan.
> - **simplify**: logical. Jika nilainya `TRUE` maka output yang dihasilkan adalah bentuk skalar.

Berikut adalah contoh penerapannya:

```
## membuat tabel frekuensi
groups <- as.factor(rbinom(32, n = 5, prob = 0.4))

tapply(groups, groups, length)
```

```
## 12 13 16
##  2  2  1
```

```
# atau
table(groups)
```

```
## groups
## 12 13 16
##  2  2  1
```

```
## membuat tabel kontingensi
# menghitung jumlah breaks berdasarkan faktor jenis wool
# dan tensi level
tapply(X=warpbreaks$breaks, INDEX=warpbreaks[,-1], FUN=sum)
```

```
##     tension
## wool   L   M   H
##    A 401 216 221
##    B 254 259 169
```

```
# menghitung mean panjang gigi babi hutan berdasarkan
# jenis suplemen dan dosisnya
tapply(ToothGrowth$len, ToothGrowth[,-1], mean)
```

```
##     dose
## supp   0.5     1     2
##   OJ 13.23 22.70 26.06
##   VC  7.98 16.77 26.14
```

```
# menghitung mpg minimum berdasarkan jumlah silinder pada mobil
tapply(mtcars$mpg, mtcars$cyl, min, simplify=FALSE)
```

```
## $`4`
## [1] 21.4
##
## $`6`
## [1] 17.8
##
## $`8`
## [1] 10.4
```

4.3 Decision Making

Decicion Making atau sering disebut sebagai *if then else statement* merupakan bentuk percabagan yang digunakan manakala kita ingin agar program dapat melakukan pengujian terhadap syarat kondisi tertentu. Pada Tabel 4.1 disajikan daftar percabangan yang digunakan pada `R`.

Table 4.1: Daftar percabangan pada `R`.

Statement	**Keterangan**
if statement	*if statement* hanya terdiri atas sebuah ekspresi *Boolean*, dan diikuti satu atau lebih *statement*
if…else statement	*if else statement* terdiri atas beberapa buah ekspresi *Boolean*. Ekspressi *Boolean* berikutnya akan dijalankan jika ekspresi *Boolan sebelumnya bernilai FALSE
switch statement	*switch statement* digunakan untuk mengevaluasi sebuah variabel beberapa pilihan

4.3.1 if statement

Pola *if statement* disajikan pada Gambar 4.2

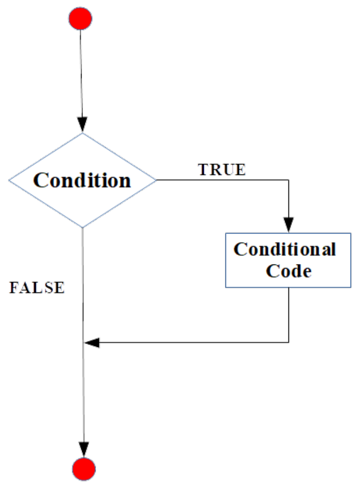

Figure 4.2: Diagram if statement (sumber: Primartha, 2018).

Berikut adalah contoh penerapan *if statement*:

```
x <- c(1:5)
if(is.vector(x)){
  print("x adalah sebuah vector")
}
```

```
## [1] "x adalah sebuah vector"
```

4.3.2 if else statement

Pola dari *if else statement* disajikan pada Gambar 4.3

Berikut adalah contoh penerapan *if else statement*:

```
x <- c("Andi","Iwan", "Adi")
if("Rina" %in% x){
  print("Rina ditemukan")
} else if("Adi" %in% x){
  print("Adi ditemukan")
} else{
  print("tidak ada yang ditemukan")
}
```

```
## [1] "Adi ditemukan"
```

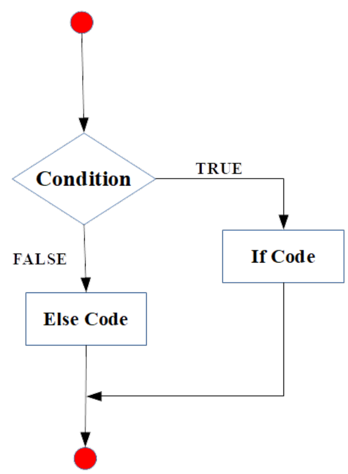

Figure 4.3: Diagram if else statement (sumber: Primartha, 2018).

4.3.3 switch statement

Pola dari *switch statement* disajikan pada Gambar 4.4

Berikut adalah contoh penerapan *switch statement*:

```
y = 3

x = switch(
  y,
  "Selamat Pagi",
  "Selamat Siang",
  "Selamat Sore",
  "Selamat Malam"
)

print(x)
```

```
## [1] "Selamat Sore"
```

4.4 Fungsi

Fungsi merupakan sekumpulan instruksi atau *statement* yang dapat melakukan tugas khusus. Sebagai contoh fungsi perkalian untuk menyelesaikan operasi perkalian, fungsi pemangkatan hanya untuk operasi pemangkatan, dll.

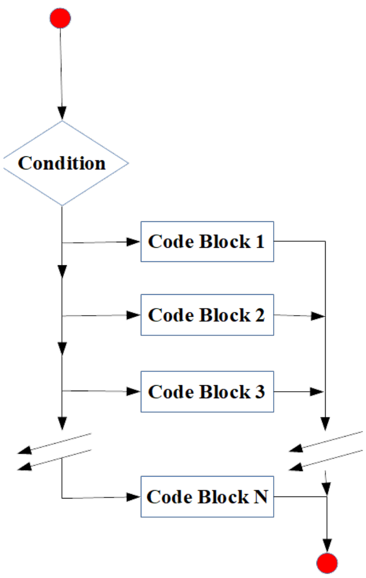

Figure 4.4: Diagram switch statement (sumber: Primartha, 2018).

Pada `R` terdapat 2 jenis fungsi, yaitu: *build in fuction* dan *user define function*. *build in fnction* merupakan fungsi bawaan `R` saat pertama kita menginstall `R`. Contohnya adalah `mean()`, `sum()`, `ls()`, `rm()`, dll. Sedangkan *user define fuction* merupakan fungsi-fungsi yang dibuat sendiri oleh pengguna.

Fungsi-fungsi buatan pengguna haruslah dideklarasikan (dibuat) terlebih dahulu sebelum dapat dijalankan. Pola pembentukan fungsi adalah sebagai berikut:

```
function_name <- function(argument_1, argument_2, ...){
  function body
}
```

Catatan:

- **function_name** : Nama dari fungsi `R`. `R` akan menyimpan fungsi tersebut sebagai objek
- **argument_1, argument_2, ...** : *Argument* bersifat opsional (tidak wajib). *Argument* dapat digunakan untuk memberi inputan kepada fungsi
- **function body** : Merupakan inti dari fungsi. Fuction body dapat terdiri atas 0 statement (kosong) hingga banyak statement.

- **return** : Fungsi ada yang memiliki *output* atau *return value* ada juga yang tidak. Jika fungsi memiliki *return value* maka *return value* dapat diproses lebih lanjut

Berikut adalah contoh penerapan *user define function*:

```
# Fungsi tanpa argument
bilang <- function(){
  print("Hello World!!")
}

# Print
bilang()
```

```
## [1] "Hello World!!"
```

```
# Fungsi dengan argumen
tambah <- function(a,b){
  print(a+b)
}

# Print
tambah(5,3)
```

```
## [1] 8
```

```
# Fungsi dengan return value
kali <- function(a,b){
  return(a*b)
}

# Print
kali(4,3)
```

```
## [1] 12
```

4.5 Debugging

Sering kali fungsi atau sintaks yang kita tulis menghasilkan error sehingga output yang kita harapkan tidak terjadi. *Debugging* merupakan langkah untuk mengecek error yang terjadi. Untuk lebih memahami proses *debugging*, berikut penulis sajikan contoh error pada suatu fungsi dapat terjadi:

```
f1 <- function(x){
  xsq <- x^2
  xsqminus4 <- xsq - 4
  print(xsqminus4)
  log(xsqminus4-4)
}

f1(6:1)
```

```
## [1] 32 21 12  5  0 -3
```

```
## Warning in log(xsqminus4 - 4): NaNs produced
```

```
## [1] 3.332 2.833 2.079 0.000   NaN   NaN
```

Untuk mengecek error yang terjadi dari sintaks tersebut, kita dapat menggunakan fungsi `debug()`. Pembaca tinggal memasukkan nama fungsi kedalam fungsi `debug()`. Fungsi tersebut akan secara otomatis akan menampilkan hasil samping dari pengaplikasian fungsi `f1()` untuk melihat sumber atau tahapan dimana error mulai muncul.

```
debug(f1)
f1(1:6)
```

```
## debugging in: f1(1:6)
## debug at <text>#1: {
##     xsq <- x^2
##     xsqminus4 <- xsq - 4
##     print(xsqminus4)
##     log(xsqminus4 - 4)
## }
## debug at <text>#2: xsq <- x^2
## debug at <text>#3: xsqminus4 <- xsq - 4
## debug at <text>#4: print(xsqminus4)
## [1] -3  0  5 12 21 32
## debug at <text>#5: log(xsqminus4 - 4)
```

```
## Warning in log(xsqminus4 - 4): NaNs produced
```

```
## exiting from: f1(1:6)
```

```
## [1]   NaN   NaN 0.000 2.079 2.833 3.332
```

Berdasarkan hasil *debugging*, `NaN` (**missing value**) muncul pada tahapan *debug* ke-4 (pembaca dapat melakukan enter terus menerus sampai proses *debug* selesai). Hal ini disebabkan karena terdapat nilai negatif pada objek `xsqminu4-4` yang selanjutnya dilakukan transformasi logaritmik. Untuk menghentikan proses *debugging* pembaca dapat mengetikkan `undebug(f1)`.

4.6 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
2. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
3. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.

Chapter 5

Pengantar Metode Numerik

Chapter ini memberikan pengantar bagi pembaca untuk mengenal terlebih dahulu mengenai metode numerik. Pada *chapter* ini akan dibahas mengenai apa itu metode numerik, perbedaannya dengan metode analitik, dan analisis error.

5.1 Mengenal Metode Numerik

Metode numerik merupakan teknik penyelesaian permsalahn yang diformulasikan secara matematis dengan menggunakan operasi hitungan (aritmatik) yaitu operasi tambah, kurang, kali, dan bagi. Metode ini digunakan karena banyak permasalahan matematis tidak dapat diselesaikan menggunakan metode analitik. Jikapun terdapat penyelesaiannya secara analitik, proses penyelesaiaannya sering kali cukup rumit dan memakan banyak waktu sehingga tidak efisien.

Terdapat keuntungan dan kerugian terkait penggunaan metode numerik. Keuntungan dari metode ini antara lain:

1. Solusi persoalan selalu dapat diperoleh.
2. Dengan bantuan komputer, perhitungan dapat dilakukan dengan cepat serta hasil yang diperoleh dapat dibuat sedekat mungkin dengan nilai sesungguhnya.
3. Tampilan hasil perhitungan dapat disimulasikan.

Adapun kelemahan metode ini antara lain:

1. Nilai yang diperoleh berupa pendekatan atau hampiran.
2. Tanpa bantuan komputer, proses perhitungan akan berlangsung lama dan berulang-ulang.

5.1.1 Perbedaan Antara Metode Numerik dan Analitik

Perbedaan antara metode numerik dan metode analitik dapat dijelaskan sebagai berikut:

1. Solusi metode numerik selalu berbentuk angka, sedangkan solusi metode analitik dapat berbentuk fungsi matematik yang selanjutnya dapat dievaluasi untuk menghasilkan nilai dalam bentuk angka.
2. Solusi dari metode numerik berupa hampiran, sedangkan metode analitik berupa solusi sejati. Kondisi ini berakibat pada nilai error metode analitik adalah 0, sedangkan metode numerik $\neq 0$.
3. Metode analitik cocok untuk permasalahan dengan model terbatas dan sederhana, sedangkan metode numerik cocok dengan semua jenis permasalahan.

5.1.2 Tahapan Penyelesaian Menggunakan Metode Numerik

Terdapat beberapa tahapan dalam menyelesaikan suatu permasalahan dengan metode numerik. Tahapan-tahapan tersebut antara lain:

- Pemodelan

Persoalan dunia nyata dimodelkan ke dalam persamaan matematika. Persamaan matematika yang terbentuk dapat berupa persamaan linier, non-linier, dan sebagainya sesuai dengan persoalan yang dihadapi.

- Penyederhanaan Model

Model matematika yang dihasilkan dari tahap 1 mungkin saja terlalu kompleks. Semakin kompleks suatu model, semakin rumit penyelesaiaannya, sehingga model perlu disederhanakan.

Seberapa sederhana model yang akan kita buat? tergantung pada permasalahan apa yang hendak pembaca selesaikan. Model yang terlalu sederhana akan tidak cocok digunakan untuk digunakan sebagai pendekatan sistem nyata atau lingkungan yang begitu kompleks. Penyederhanaan dapat berupa asumsi sejumlah variabel yang terlibat tidak signifikan, atau asumsi kondisi reaktor (*steady* atau *non-steady*).

- Formulasi Numerik

Setelah model matematika sederhana diperoleh, tahap selanjutnya adalah memformulasikan model matematika secara numerik. Tahapan ini terdiri atas: + menentukan metode numerik yang akan dipakai bersama-sama dengan analisis galat (error) awal. + menyusun algoritma dari metode numerik yang dipilih.

- Pemrograman

Tahap selanjutnya adalah menerjemahkan algoritma ke dalam program komputer. Pada tahapan ini pembaca bisa memilih bahasa pemrograman yang pembaca kuasai.

Dalam buku ini kita hanya akan berfokus pada bahasa pemrograman `R`. Pembaca dapat menggunakan bahasa pemrograman lain selain dari buku ini. Pembaca hanya perlu memperhatikan bagaimana penulis membangun algoritma penyelesaian dan memtransfernya menjadi bentuk sintaks `R`. Dari sintaks tersebut pembaca dapat melihat bagaimana meletakakkan tiap tahapan algoritma menjadi sintaks pada bahasa pemrograman.

- Operasional

Sebelum digunakan dengan data sesungguhnya, program komputer perlu dilakukan uji coba dengan data simulasi dan dievaluasi hasilnya. jika hasil keluaran diyakini sudah sesuai, baru dioperasikan dengan data yang sesungguhnya.

- Evaluasi

Bila program sudah selesai dijalankan dengan data yang sesungguhnya, maka hasil yang diperoleh dilakukan interpretasi, meliputi analisis hasil keluaran dan membandingkannya dengan prinsip dasar dan hasil-hasil empriik untuk menaksir kualitas soluasi numerik termasuk keputusan untuk menjalankan kembali progrma dengan memperoleh hasil yang lebih baik.

5.2 Akurasi dan Presisi

Perhatikan Gambar 5.1 berikut:

Pada Gambar 5.1 terdapat 4 buah kondisi ketika kita menembakkan beberapa perluru pada sebuah sasaran. Tujuan kita disini adalah untuk menembak bagian tengah sasaran tersebut.

Pada Gambar (a) dan (c) pada Gambar 5.1 merupakan gambar yang menunjukkan seseorang telah berhasil mengenai bagian tengah sasaran tersebut dapat kita katakan pula tembakan pada kedua gambar tersebut akurat. Akurat dalam hal ini dapat diartikan suatu kondisi dimana kedekatan lubang peluru dengan

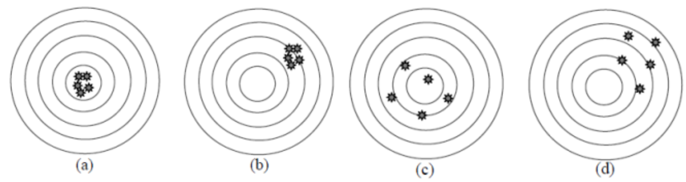

Figure 5.1: 4 ilustrasi terkait akurasi dan presisi

pusat sasaran. Secara umum akurasi diartikan sebagai tingkat kedekatan pengukuran kuantitas terhadap nilai sebenarnya.

Terdapat dua buah cara untuk mengukur akurasi. Metode pengukuran akurasi antara lain: error absolut dan error relatif.

Error absolut merupakan nilai absolut dari selisih antara nilai sebenarnya x dengan nilai observasi x'. Error absolut dapat dituliskan menggunakan Persamaan (5.1).

$$\epsilon_A = |x - x'| \tag{5.1}$$

Pengukuran lain yang sering digunakan untuk mengukur akurasi adalah error relatif. Berbeda dengan error absolut, error relatif membagi selisih antara nilai sebenarnya x dan nilai observasi x' dengan nilai sebenarnya. Hasil yang diperoleh merupakan nilai tanpa satuan. Persamaan error relatif disajikan pada Persamaan (5.2).

$$\epsilon_R = \left| \frac{x - x'}{x} \right| \tag{5.2}$$

Dalam suatu pengukuran, hal lain yang perlu diperhatikan selain akurasi adalah presisi. Presisi adalah sejauh mana pengulangan pengukuran dalam kondisi yang tidak berubah mendapat hasil yang sama. Berdasarkan Gambar 5.1, Gambar (a) dan (b) menunjukkan kepresisian yang tinggi. Hal ini terlihat dari jarak antara lubang peluru yang saling berdekatan dan mengelompok.

Berdasarkan Gambar 5.1 dapat kita simpulkan bahwa dalam suatu sistem pengukuran akan terdapat 4 buah kondisi. Pengukuran akurat dan presisi (Gambar (a)), tidak akurat namun presisi (Gambar (b)), akurat namun tidak presisi (Gambar (c)), dan tidak akurat serta tidak presisi (Gambar (d)).

Dari kondisi-kondisi tersebut, akan meuncul yang dinamakan error. Dalam analisa numerik error atau kesalahan menjadi hal yang perlu diperhatikan.

5.3 Error Numerik

Kesalahan numerik merupakan error atau kesalahan yang timbul akibat adanya proses pendekatan atau hampiran. Kesalahan numerik terjadi karena tiga hal, antara lain:

- **Kesalahan bawaan (*inherent error*)**, merupakan kesalahan data yang timbul akibat adanya pengkuran, *human error* seperti kesalahan pencatatan, atau tidak memahami hukum-hukum fisik dari data yang diukur.
- **Kesalahan pembulatan (*round-off error*)**, adalah kesalahan yang terjadi karena adanya pembulatan. Contoh: 3,142857143... menjadi 3,14.
- **Kesalahan pemotongan (*truncation error*)**, adalah kesalahan yang ditimbulkan pada saat dilakukan pengurangan jumlah angka signifikan.

Kesalahan atau error dapat diukur menggunakan Persamaan (5.1) dan Persamaan (5.2) yang telah penulis jelaskan pada Chapter 5.2.

5.4 Referensi

1. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
2. Sidiq, M. Tanpa Tahun. **Materi Kuliah Metode Numerik**. Repository Universitas Dian Nuswantoro.
3. Subakti, I. 2006. **Metode Numerik**. Institut Teknologi Sepuluh Nopember.
4. Sutarno,H., Rachmatin,D. 2008. **Hands Out Metode Numerik**. Universitas Pendidikan Indonesia.

Chapter 6

Aljabar Linier

Pada *chapter* ini penulis akan menjelaskan mengenai cara untuk menyelesaikan sistem persamaan linier. Adapun yang akan dibahas pada *chapter* ini antara lain:

- operasi Vektor dan matriks
- Metode Eliminasi Gauss
- Metode Dekomposisi matriks
- Studi Kasus

6.1 Vektor dan matriks

Pada Chapter 2.7 dan Chapter 2.8 telah dijelaskan sekilas bagaimana cara melakukan operasi pada vektor dan matriks. Pada *chapter* ini, penulis akan menambahkan operasi-operasi lain yang dapat dilakukan pada vektor dan matriks. Dasar-dasar operasi ini selanjutnya akan digunakan sebagai dasar menyusun algoritma penyelesaian sistem persamaan linier.

6.1.1 Operasi Vektor

Misalkan saja diberikan vektor u dan v yang ditunjukkan pada Persamaan (6.1).

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \; dan \; v \; = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{6.1}$$

Jika kita menambahkan atau mengurangkan nilai elemen vektor dengan suatu skalar (konstanta yang hanya memiliki besaran), maka operasi penjumlahan/pengurangan akan dilakukan pada setiap elemen vektor.

$$u \pm x = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \pm x = \begin{bmatrix} u_1 \pm x \\ u_2 \pm x \\ \vdots \\ u_n \pm x \end{bmatrix} \tag{6.2}$$

Jika kita melakukan penjumlahan pada vektor u dan v, maka operasi akan terjadi pada masing-masing elemen dengan indeks yang sama.

$$u \pm v = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \pm v \ = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 \pm v_1 \\ u_2 \pm v_2 \\ \vdots \\ u_n \pm v_n \end{bmatrix} \tag{6.3}$$

Untuk lebih memahami operasi tersebut, berikut penulis berikan contoh penerapannya pada `R`:

```
u <- seq(1,5)
v <- seq(6,10)

# penjumlahan
u+v
```

```
## [1]  7  9 11 13 15
```

```
# penguranga
u-v
```

```
## [1] -5 -5 -5 -5 -5
```

Bagaimana jika kita melakukan operasi dua vektor, dimaana salah satu vektor memiliki penjang yang berbeda?. Untuk memnjawab hal tersebut, perhatikan sintaks berikut:

```
x <- seq(1,2)
u+x
```

```
## Warning in u + x: longer object length is not a
## multiple of shorter object length
```

```
## [1] 2 4 4 6 6
```

Berdasarkan contoh tersebut, `R` akan mengeluarkan peringatan yang menunjukkan operasi dilakukan pada vektor dengan panjang berbeda. `R` akan tetap melakukan perhitungan dengan menjumlahkan kembali vektor u yang belum dijumlahkan dengan vektor x sampai seluruh elemen vektor u dilakukan operasi penjumlahan.

Operasi lain yang dapat dilakukan pada vektor adalah menghitung *inner product* dan panjang vektor. Inner product dihitung menggunakan Persamaan (6.4).

$$u.v = \sum_{i=1}^{n} u_1v_1 + u_2v_2 + \cdots + u_nv_n \tag{6.4}$$

Panjang vektor atau vektor yang telah dinormalisasi dihitung menggunakan Persamaan (6.5)

$$|u| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2} \tag{6.5}$$

Berikut adalah contoh bagaimana cara menghitung `inner product` dan panjang vektor menggunakan `R`:

```r
# inner product
u%*%v
```

```
##      [,1]
## [1,]  130
```

```r
# panjang vektor u
sqrt(sum(u*u))
```

```
## [1] 7.416
```

6.1.2 Operasi matriks

Misalkan kita memiliki 2 buah matriks A dan B.

$$A = \begin{bmatrix} a_{1.1} & a_{1.2} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & \cdots & a_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & \cdots & a_{m.n} \end{bmatrix} \; dan \; B = \begin{bmatrix} b_{1.1} & b_{1.2} & \cdots & b_{1.n} \\ b_{2.1} & b_{2.2} & \cdots & b_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m.1} & b_{m.2} & \cdots & b_{m.n} \end{bmatrix} \tag{6.6}$$

Jika salah satu matriks tersebut dijumlahkan atau dikurangkan dengan skalar.

$$A \pm x = \begin{bmatrix} a_{1.1} & a_{1.2} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & \cdots & a_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & \cdots & a_{m.n} \end{bmatrix} \pm x = \begin{bmatrix} a_{1.1} \pm x & a_{1.2} \pm x & \cdots & a_{1.n} \pm x \\ a_{2.1} \pm x & a_{2.2} \pm x & \cdots & a_{2.n} \pm x \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} \pm x & a_{m.2} \pm x & \cdots & a_{m.n} \pm x \end{bmatrix} \tag{6.7}$$

Jika kedua matriks A dan B saling dijumlahkan atau dikurangkan. Perlu diperhatikan bahwa penjumlahan dua buah matriks hanya dapat dilakukan pada matriks dengan ukuran yang seragam.

$$\begin{aligned} A \pm B &= \begin{bmatrix} a_{1.1} & a_{1.2} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & \cdots & a_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & \cdots & a_{m.n} \end{bmatrix} \pm \begin{bmatrix} b_{1.1} & b_{1.2} & \cdots & b_{1.n} \\ b_{2.1} & b_{2.2} & \cdots & b_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m.1} & b_{m.2} & \cdots & b_{m.n} \end{bmatrix} \\ &= \begin{bmatrix} a_{1.1} \pm b_{1.1} & a_{1.2} \pm b_{1.2} & \cdots & a_{1.n} \pm b_{1.n} \\ a_{2.1} \pm b_{2.1} & a_{2.2} \pm b_{2.2} & \cdots & a_{2.n} \pm b_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} \pm b_{m.1} & a_{m.2} \pm b_{m.2} & \cdots & a_{m.n} \pm b_{m.n} \end{bmatrix} \end{aligned} \tag{6.8}$$

Untuk lebih memahaminya, berikut disajikan contoh operasi penjumlahan pada matriks:

```r
A <- matrix(1:9,3)
B <- matrix(10:18,3)
C <- matrix(1:6,3)

# penjumlahan dengan skalar
A+1
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    8
## [2,]    3    6    9
## [3,]    4    7   10
```

```r
# penjumlahan A+B
A+B
```

```
##      [,1] [,2] [,3]
```

```
## [1,]   11   17   23
## [2,]   13   19   25
## [3,]   15   21   27
```

```
# penjumlahan
A+C
```

Operasi pehitungan lain yang penting pada matriks adalah operasi perkalian matriks. Perlu diperhatikan bahwa untuk perkalian matriks, jumlah kolom matriks sebelah kiri harus sama dengan jumlah baris pada matriks sebelah kanan. Perkalian antara dua matriks disajikan pada Persamaan (6.9).

$$A_{m.n} \times B_{n.r} = AB_{m.r} \tag{6.9}$$

Pada `R` perkalian matriks dilakukan menggunakan operator `%*%`. Berikut adalah contoh perkalian matriks pada `R`:

```
# Perkalian matriks
A%*%B
```

```
##      [,1] [,2] [,3]
## [1,]  138  174  210
## [2,]  171  216  261
## [3,]  204  258  312
```

6.2 Operasi Baris Elementer

Terdapat tiga buah operasi dasar pada baris matriksoperasi baris elementer. Ketiga operasi ini akan menjadi dasar operasi *sub-chapter* selanjutnya. Ketiga operasi dasar tersebut antara lain:

1. ***Row Scalling***. Mengalikan baris matriks dengan konstanta bukan nol.
2. ***Row Swaping***. Menukar urutan baris pada sebuah matriks (contoh: menukar baris 1 dengan baris 2 dan sebaliknya).
3. ***Row Replacement***. Baris matriks diganti dengan hasil penjumlahan atau pengurangan baris matriks tersebut dengan baris matriks lainnya, dimana baris matriks lainnya yang akan dijumlahkan/dikurangkan dengan matriks tersebut telah dilakukan proses *row scalling*. Luaran yang diperoleh pada umumnya adalah nilai nol pada baris matriks awal atau akhir.

Ketiga proses tersebut akan terjadi secara berulang, khusunya jika kita hendak mengerjakan sistem persamaan linier menggunakan algoritma eliminasi Gauss. Untuk mempermudah proses tersebut, kita dapat membuat masing-masing fungsi untuk masing-masing operasi tersebut. Algoritma fungsi-fungsi tersebut selanjutnya menjadi dasar penyusunan algoritma fungsi-fungsi eliminasi Gauss dan dekomposisi matriks yang akan dijelaskan pada *chapter* selanjutnya.

Fungsi *row scalling* pada `R` dapat dituliskan pada sintaks berikut:

```
scale_row <- function(m, row, k){
 m[row, ] <- m[row, ]*k
 return(m)
}
```

Berikut adalah contoh penerapannya:

```
# membuat matriks A
(A <- matrix(1:15, nrow=5))
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
# lakukan scaling pada row 2 dengan nilai 10
scale_row(m=A, row=2, 10)
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]   20   70  120
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

> **Catatan:** Untuk menyimpan hasil perhitungan, simpan proses perhitungan dalam sebuah objek (lihat Chapter 2.5).

Row swapping merupakan proses yang berulang, kita perlu menyimpan terlebih dahulu baris matriks pertama kedalam sebuah objek. Baris matriks pertama selanjutnya diganti dengan baris matriks kedua, sedangkan baris matriks kedua selanjutnya akan diganti dengan baris matriks pertama yang telah terlebih dahulu disimpan dalam sebuah objek. Fungsi *row swapping* pada `R` dapat dituliskan pada sintaks berikut:

```r
swap_row <- function(m, row1, row2){
  row_tmp <- m[row1, ]
  m[row1, ] <- m[row2, ]
  m[row2, ] <- row_tmp
  return(m)
}
```

Berikut merupakan contoh penerapan fungsi `swap_row()`:

```r
# pertukarkan baris 2 dengan baris 5
swap_row(m=A, row1=2, row2=5)
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    5   10   15
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    2    7   12
```

Pada proses *row replacement*, proses perhitungan dilakukan dengan melakukan penjumlahan suatu baris matriks dengan baris matriks lainnya dengan terlebih dahulu melakukan *row scalling* terhadap matriks lainnya. Berikut adalah fungsi `replace_row()` yang ditulis pada `R`:

```r
replace_row <- function(m, row1, row2, k){
  m[row2, ] <- m[row2, ] + m[row1, ]*k
  return(m)
}
```

Berikut adalah contoh penerapan fungsi `replace_row()`:

```r
replace_row(m=A, row1=1, row2=3, k=-3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    0  -10  -20
## [4,]    4    9   14
## [5,]    5   10   15
```

6.3 Eliminasi Gauss

Pada *sub-chapter* ini kita akan menggunakan operasi baris elementer yang telah dijelaskan pada Chapter 2.5. Terdapat dua topik yang akan dibahas pada *sub-chapter* ini, yaitu: *row echelon form* termasuk *reduced row echelon form* dan matriks tridiagonal.

Eliminasi Gauss merupakan sebuah cara untuk mencari penyelesaian sistem persamaan linier. Ide dasar dari eliminasi Gauss adalah melakukan operasi matematika pada baris matriks (lihat Chapter 2.5) dan melanjutkannya sampai hanya tersisa satu variabel saja. Kita dapat melakukan lebih dari satu operasi baris elementer pada proses elmininasi ini (contoh: mengalikan sebuah baris dengan konstanta dan menjumlahkan hasilnya pada baris lain).

6.3.1 *Row Echelon Form*

Sebuah matriks merupakan *row echelon form* jika matriks tersebut memenuhi beberapa kondisi:

1. Angka bukan nol pertama dari kiri (*leading coefficient*) selalu di sebelah kanan angka bukan nol pertama pada baris di atasnya.
2. Baris yang terdiri dari semua nol ada di bagian bawah matriks.

Misalkan terdapat persamaan linier seperti yang ditunjukkan pada Persamaan (6.10).

$$\begin{gathered}
a_{1.1}x_1 + a_{1.2}x_2 + a_{1.3}x_3 + \cdots + a_{1.n}x_n = b_1 \\
a_{2.1}x_1 + a_{2.2}x_2 + a_{2.3}x_3 + \cdots + a_{2.n}x_n = b_2 \\
a_{3.1}x_1 + a_{3.2}x_2 + a_{3.3}x_3 + \cdots + a_{3.n}x_n = b_3 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
a_{m.1}x_1 + a_{m.2}x_2 + a_{m.3}x_3 + \cdots + a_{m.n}x_n = b_n
\end{gathered} \tag{6.10}$$

dimana $a_{i.j}$ untuk $i = 1$ sampai dengan m dan $j = 1$ sampai dengan n merupakan koefisien persamaan linier. x_i untuk $i = 1$ sampai dengan n merupakan variabel bebas pada sistem persamaan linier.

Persamaan linier pada Persamaan (6.10) dapat dinyatakan ke dalam bentuk matriks pada Persamaan (6.11).

$$\begin{bmatrix} a_{1.1} & a_{1.2} & a_{1.3} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & a_{2.3} & \cdots & a_{2.n} \\ a_{3.1} & a_{3.2} & a_{3.3} & \cdots & a_{3.n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & a_{m.3} & \cdots & a_{m.n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.11}$$

$$AX = B \tag{6.12}$$

dimana:

- matriks A merupakan matriks koefisien / Jacobian
- vaktor X merupakan vaktor variabel
- vektor B merupakan vektor konstanta

matriks pada Persamaan (6.11) dapat diubah menjadi *augmented matrix*, yaitu: perluasan matriks A dengan menambahkan vektor B pada kolom terakhirnya.

$$\begin{bmatrix} a_{1.1} & a_{1.2} & a_{1.3} & \cdots & a_{1.n} & b_1 \\ a_{2.1} & a_{2.2} & a_{2.3} & \cdots & a_{2.n} & b_2 \\ a_{3.1} & a_{3.2} & a_{3.3} & \cdots & a_{3.n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{m.1} & a_{m.2} & a_{m.3} & \cdots & a_{m.n} & b_n \end{bmatrix} \tag{6.13}$$

$$A = [A|B] \tag{6.14}$$

Teorema 6.1 (spltheorem)**.** *Suatu sistem persamaan linier mempunyai penyelesaian tunggal bila memenuhi syarat-syarat sebagai berikut:*

- ukuran persamaan linier simultan bujursangkar (jumlah persamaan sama dengan jumlah variabel bebas).
- sistem persamaan linier *non-homogen* di mana minimal ada satu nilai vektor konstanta B tidak nol atau terdapat $b_n \neq 0$.
- Determinan dari matriks koefisiensistem persamaan linier tidak sama dengan nol.

Untuk memperoleh penyelesaian sistem persamaan linier, Persamaan (6.13) perlu dilakukan operasi baris elementer. Hasil operasi baris dasar akan menghasilkan matriks *row echelon form* yang disajikan pada Persamaan (6.15).

$$\begin{bmatrix} a_{1.1} & a_{1.2} & a_{1.3} & \cdots & a_{1.n} & b_1 \\ a_{2.1} & a_{2.2} & a_{2.3} & \cdots & a_{2.n} & b_2 \\ a_{3.1} & a_{3.2} & a_{3.3} & \cdots & a_{3.n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{m.1} & a_{m.2} & a_{m.3} & \cdots & a_{m.n} & b_n \end{bmatrix} \Longrightarrow \begin{bmatrix} c_{1.1} & c_{1.2} & c_{1.3} & \cdots & c_{1.n} & d_1 \\ 0 & c_{2.2} & c_{2.3} & \cdots & c_{2.n} & d_2 \\ 0 & 0 & c_{3.3} & \cdots & c_{3.n} & d_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & c_{m.n} & d_n \end{bmatrix} \tag{6.15}$$

Sehingga penyelesaian sistem persamaan linier dapat diperoleh menggunakan Persamaan (6.16).

$$\begin{aligned} x_n &= \tfrac{d_n}{c_{m.n}} \\ x_{n-1} &= \tfrac{1}{C_{m-1.n-1}} \left(d_{n-1} - c_{m-1.n} x_n\right) \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_2 &= \tfrac{1}{c_{2.2}} \left(d_2 - c_{2.3} x_3 - c_{2.4} x_4 - \cdots - c_{2.n} x_n\right) \\ x_1 &= \tfrac{1}{c_{1.1}} \left(d_1 - c_{1.2} x_2 - c_{1.3} x_3 - \cdots - c_{1.n} x_n\right) \end{aligned} \tag{6.16}$$

Contoh 6.1. Selesaikan sistem persamaan berikut:

$$\begin{aligned} x_1 + x_2 + x_3 &= 6 \\ x_1 + 2x_2 - x_3 &= 2 \\ 2x_1 + x_2 + 2x_3 &= 10 \end{aligned}$$

Jawab:

Augmented matrix sistem persamaan linier tersebut adalah sebagai berikut:

$$\begin{bmatrix} 1 & 1 & 1 & 6 \\ 1 & 2 & -1 & 2 \\ 2 & 1 & 2 & 10 \end{bmatrix}$$

Operasi baris elementer selanjutnya dilakukan pada matriks tersebut. Pada langkah pertama, baris ke-2 dikurangkan dengan baris ke-1 ($B_2 - B_1$) dan baris ke-3 dikurangkan oleh dua kali baris ke-1 ($B_3 - 2B_1$).

$$\begin{bmatrix} 1 & 1 & 1 & 6 \\ 1 & 2 & -1 & 2 \\ 2 & 1 & 2 & 10 \end{bmatrix} \underset{B_3 - 2B_1}{\overset{B_2 - B_1}{\Longrightarrow}} \begin{bmatrix} 1 & 1 & 1 & 6 \\ 0 & 1 & -2 & -4 \\ 0 & -1 & 0 & -2 \end{bmatrix}$$

Hasil dari langkah pertama tersebut, selanjutnya menjadi input dari langkah selanjutnya. Pada langkah selanjutnya operasi baris elementer kembali dilanjutkan. Baris ke-3 dikurangkan denganbaris ke-2 ($B_3 - B_2$).

$$\begin{bmatrix} 1 & 1 & 1 & 6 \\ 0 & 1 & -2 & -4 \\ 0 & -1 & 0 & -2 \end{bmatrix} \overset{B_3 - B_2}{\Longrightarrow} \begin{bmatrix} 1 & 1 & 1 & 6 \\ 0 & 1 & -2 & -4 \\ 0 & 0 & -2 & -6 \end{bmatrix}$$

Setelah diperoleh matriks *row echelon form* selanjutnya penyelesaian persamaan dapat dikerjakan menggunakan Persamaan (6.16).

$$x_3 = \frac{-6}{-2} = 3$$
$$x_2 = \frac{1}{1}(-4 - (2)\,3) = 2$$
$$x_1 = \frac{1}{1}(6 - 2 - 3) = 1$$

Algoritma Row Echelon Form

1. Masukkan matriks A, dan vektor B beserta ukurannya n
2. Buat *augmented matrix* $[A|B]$ namakan dengan A
3. Untuk baris ke-i dimana $i = 1$ s/d n, perhatikan apakah nilai $a_{i,j}$ sama dengan nol. **a) Bila iya**, lakukan *row swapping* antara baris ke-i dan baris ke-$i + k \leq n$, dimana $a_{i+k,j}$ tidak sama dengan nol. Bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian, **b) Bila tidak**, lanjutkan.
4. Untuk baris ke-j, dimana $j = i + 1$ s/d n, lakukan operasi baris elementer:**a)** Hitung $c = \frac{a_{j,i}}{a_{i,i}}$, **b)** untuk kolom k, dimana $k = 1$ s/d $n + 1$, hitung $a_{j,k} = a_{j,k} - c.a_{i,k}$.
5. Hitung akar, untuk $i = n$ s/d 1 (bergerak dari baris pertama) menggunakan Persamaan (6.16).

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi pada `R` untuk menyelesaikan sistem persamaan linier menggunakan matriks *row echelon form*. Fungsi yang akan dibentuk hanya sampai pada algoritma ke-4. Proses substitusi akan dilakukan secara manual. Berikut adalah sintaks yang digunakan:

```
ref_matrix <- function(a){
  m <- nrow(a)
  n <- ncol(a)
  piv <- 1

# cek elemen diagonal apakah bernilai nol
  for(row_curr in 1:m){
    if(piv <= n){
      i <- row_curr
      while(a[i, piv] == 0 && i < m){
        i <- i+1
        if(i > m){
          i <- row_curr
          piv <- piv+1
          if(piv > n)
            return(a)
```

```
        }
      }

# jika diagonal bernilai nol, lakukan row swapping
    if(i != row_curr)
      a <- swap_row(a, i, row_curr)

# proses triangulasi untuk membentuk matriks segitiga atas
    for(j in row_curr:m)
      if(j != row_curr){
        c <- a[j, piv]/a[row_curr, piv]
        a <- replace_row(a, row_curr, j, -c)
      }
    piv <- piv+1
    }
  }
  return(a)
}
```

Dengan menggunakan fungsi `ref_matrix()`, kita dapat membentuk matriks *row echelon form* pada Contoh 6.1.

```
am <- c(1,1,2,
        1,2,1,
        1,-1,2,
        6,2,10)
(m <- matrix(am, nrow=3))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    6
## [2,]    1    2   -1    2
## [3,]    2    1    2   10
```

```
ref_matrix(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    6
## [2,]    0    1   -2   -4
## [3,]    0    0   -2   -6
```

matriks yang diperoleh selanjutnya dapat diselesaikan menggunakan Persamaan (6.16).

Contoh 6.2. Dengan menggunakan fungsi `ref_matrix()`, buatlah matriks *row echelon form* dari sistem persamaan linier berikut:

$$
\begin{aligned}
2x_1 + x_2 - x_3 &= 1 \\
3x_1 + 2x_2 - 2x_3 &= 1 \\
x_1 - 5x_2 + 4x_3 &= 3
\end{aligned}
$$

Jawab:

Augmented matrix dari sistem persamaan tersebut adalah sebagai berikut:

$$
\begin{bmatrix}
2 & 1 & -1 & 1 \\
3 & 2 & -2 & 1 \\
1 & -5 & 4 & 3
\end{bmatrix}
$$

Penyelesaian matriks tersebut adalah sebagai berikut:

```
(m <- matrix(c(2,3,1,
               1,2,-5,
               -1,-2,4,
               1,1,3), nrow=3))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    1   -1    1
## [2,]    3    2   -2    1
## [3,]    1   -5    4    3
```

```
ref_matrix(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2  1.0 -1.0  1.0
## [2,]    0  0.5 -0.5 -0.5
## [3,]    0  0.0 -1.0 -3.0
```

Proses lebih lanjut akan menghasilkan penyelesaian sebagai berikut:

$$
\begin{aligned}
x_1 &= 1 \\
x_2 &= 2 \\
x_3 &= 3
\end{aligned}
$$

6.3.2 Eliminasi Gauss-Jordan

Berbeda dengan metode eliminasi Gauss yang telah dijelaskan pada Chapter 6.3.1, metode eliminasi Gauss-Jordan membentuk matriks menjadi bentuk *reduced row echelon form.* Metode ini merupakan pengembangan metode eliminasi

Gauss, dimana matriks sebelah kiri *augmented matrix* diubah menjadi matriks diagonal (lihat Persamaan (6.17)).

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} & b_n \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & d_1 \\ 0 & 1 & 0 & \cdots & 0 & d_2 \\ 0 & 0 & 1 & \cdots & 0 & d_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & d_n \end{bmatrix} \tag{6.17}$$

Sehingga penyelesaian persamaan linier tersebut adalah nilai $d_1, d_2, d3, \ldots, d_n$ dan atau:

$$\begin{gathered} x_1 = d_1 \\ x_2 = d_2 \\ x_3 = d_3 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n = d_n \end{gathered} \tag{6.18}$$

Contoh 6.3. Selesaikan sistem persamaan berikut:

$$\begin{gathered} x_1 + x_2 = 3 \\ 2x_1 + 4x_2 = 8 \end{gathered}$$

Jawab:

Augmented matrix dari persamaan linier tersebut adalah sebagai berikut:

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 4 & 8 \end{bmatrix}$$

Operasi baris elementer selanjutnya dilakukan pada matriks tersebut.

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 4 & 8 \end{bmatrix} \overset{B_2 - 2B_1}{\Longrightarrow} \begin{bmatrix} 1 & 1 & 3 \\ 0 & 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 0 & 2 & 2 \end{bmatrix} \overset{\frac{B_2}{2}}{\Longrightarrow} \begin{bmatrix} 1 & 1 & 3 \\ 0 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 0 & 1 & 1 \end{bmatrix} \overset{B_1 - B_2}{\Longrightarrow} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

Penyelesaian persamaan linier tersebut adalah sebagai berikut:

$$x_1 = 2 \; dan \; x_2 = 1$$

Algoritma Metode Eliminasi Gauss-Jordan

1. Masukkan matriks A dan vektor B beserta ukurannya n
2. Buat *augmented matrix* $[A|B]$ namakan dengan A
3. Untuk baris ke-i dimana $i = 1$ s/d n

- Perhatikan apakah nilai $a_{i.i}$ sama dengan nol:
  - **Bila ya**: pertukakan baris ke-i dan baris ke-$i+k \leq n$, dimana $a_{i+k.i}$ tidak sama dengan nol, bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian.
  - **Bila tidak**: lanjutkan
- Jadikan nilai diagonalnya menjadi satu dengan cara untuk setiap kolom k dimana $k = 1$ s/d $n+1$, hitung $a_{i.k} = \frac{a_{i.k}}{a_{i.i}}$

4. Untuk baris ke-j, dimana $j = i+1$ s/d n. Lakukan operasi baris elementer untuk kolom k dimana $k = 1$ s/d n.

- Hitung $c = a_{j.i}$
- Hitung $a_{j.k} = a_{j.k} - c.a_{i.k}$

5. Penyelesaian untuk $i = n$ s/d 1 disajikan pada Persamaan (6.18).

Dari algoritma tersebut, kita dapat membangun sebuah fungsi menggunakan `R`. Fungsi tersebut adalah sebagai berikut:

```
gauss_jordan <- function (a){
    m <- nrow (a)
    n <- ncol (a)
    piv <- 1

# cek elemen diagonal utama apakah bernilai nol
    for(row_curr in 1:m){
        if(piv <= n){
```

```
            i <- row_curr
            while(a[i, piv] == 0 && i < m){
                i <- i + 1
                if(i > m){
                    i <- row_curr
                    piv <- piv + 1
                    if(piv > n)
                        return (a)
                }
            }

# jika diagonal utama bernilai nol,lakukan row swapping
            if(i != row_curr)
                a <- swap_row(a, i, row_curr)

# proses pembentukan matriks reduced row echelon form
            piv_val <- a[row_curr , piv]
            a <- scale_row (a, row_curr , 1 / piv_val)
            for(j in 1: m){
                if(j != row_curr){
                    k <- a[j, piv]/a[row_curr, piv]
                    a <- replace_row (a, row_curr, j, -k)
                }
            }
            piv <- piv + 1
        }
    }
    return (a)
}
```

Dengan menggunakan fungsi `gauss_jordan()`, sistem persamaan linier pada Contoh 6.3:

```
(m <- matrix(c(1,2,1,4,3,8), nrow=2))
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    3
## [2,]    2    4    8
```

```
gauss_jordan(m)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    2
## [2,]    0    1    1
```

Contoh 6.4. Dengan menggunakan fungsi `gauss_jordan()`, carilah penyelesaian sistem persamaan linier pada Contoh 6.1 dan Contoh 6.2:

Jawab:

Untuk Contoh 6.1:

```
am <- c(1,1,2,
        1,2,1,
        1,-1,2,
        6,2,10)
m <- matrix(am, nrow=3)

gauss_jordan(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    1
## [2,]    0    1    0    2
## [3,]    0    0    1    3
```

Untuk Contoh 6.2:

```
m <- matrix(c(2,3,1,1,2,-5,
              -1,-2,4,1,1,3),
            nrow=3)
gauss_jordan(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    1
## [2,]    0    1    0    2
## [3,]    0    0    1    3
```

6.3.3 Matrik Tridiagonal

Metode eliminasi Gauss merupakan metode yang sederhana untuk digunakan khususnya jika semua koefisien bukan nol berkumpul pada diagonal utama dan beberapa diagonal sekitarnya. Suatu sistem yang bersifat demikian disebut sebagai *banded* dan banyaknya diagonal yang memuat koefisien bukan nol disebut sebagai *bandwidth*. Contoh khusus yang sering dijumpai adalah matriks tridiagonal yang memiliki *bandwidth* tiga.

Proses eliminasi untuk matriks tridiagonal bersifat trivial karena dengan membentuk sebuah subdiagonal tambahan, proses substitusi mundur segera dapat dilakukan. Bentuk matriks tridiagonal disajikan pada Persamaan (6.19).

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & 0 \\ 0 & a_{3,2} & a_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.19}$$

Penyelesaian persamaan tersebut disajikan pada Persamaan (6.20).

$$x_n = \frac{b_n}{a_{m.n}};\ x_i = \frac{b_i - a_{i,j+1}x_{i+1}}{a_{i,j}} \tag{6.20}$$

dimana $i = n-1, n-2, ..., 1$.

Pada beberapa *textbook*, diagonal matriks sering dilambangkan dengan l(diagonal bawah), d(diagonal tengah), dan u (diagonal atas). Bentuk matriksnya disajikan pada Persamaan (6.21).

$$\begin{bmatrix} d_1 & u_2 & 0 & \cdots & 0 \\ l_2 & d_2 & u_3 & \cdots & 0 \\ 0 & l_3 & d_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.21}$$

Algoritma Penyelesaian Matrik Tridiagonal

1. Bentuk sistem persamaan linier menjadi matriks pada Persamaan (6.21).
2. Lakukan *foward sweep*. Setiap elemen diagonal l dieliminasi menggunakan reduksi baris.

- Untuk $i = 1$
  - Hitung $u_1 = \frac{u_1}{d_1}$
  - Hitung $b_1 = \frac{b_1}{d_1}$
- Untuk $i = 2$ s/d $n-1$
  - Hitung $u_i = \frac{u_i}{d_i - l_i \times u_{i-1}}$
  - Hitung $b_i = \frac{b_i - l_i \times u_{i-1}}{d_i - l_i \times u_{i-1}}$
- Hitung $b_n = \frac{b_n - l_n \times u_{n-1}}{d_n - l_n \times u_{n-1}}$

3. Lakukan *backward sweep*. Setiap elemen diagonal u dilakukan eliminasi.

- Untuk $i = n - 1$ s/d 1
  - Hitung $x_n = b_i - u_i \times x_{i+1}$
- Hitung $x_n = b_n$

Berdasarkan algoritma tersebut, kita dapat membangun sebuah fungsi pada R. Fungsi penyelesaian matriks tridiagonal disajikan sebagai berikut:

```
tridiagmatrix <- function (L, D, U, b){
  n <- length (D)
  L <- c(NA , L)

  ## forward sweep
  U[1] <- U[1] / D[1]
  b[1] <- b[1] / D[1]
  for(i in 2:(n - 1)){
      U[i] <- U[i] / (D[i] - L[i] * U[i - 1])
      b[i] <- (b[i] - L[i] * b[i - 1]) /
      (D[i] - L[i] * U[i - 1])
  }
  b[n] <- (b[n] - L[n] * b[n - 1])/(D[n] - L[n] * U[n - 1])

  ## backward sweep
  x <- rep.int (0, n)
  x[n] <- b[n]
  for(i in (n - 1) :1)
      x[i] <- b[i] - U[i] * x[i + 1]
  return (x)
}
```

Contoh 6.5. Selesaikan sistem persamaan berikut menggunakan fungsi `tridiagmatrix()` dan fungsi `gauss_jordan()`!

$$\begin{aligned} 3x_1 + 4x_2 &= 20 \\ 4x_1 + 5x_2 - 2x_3 &= 28 \\ 2x_2 + 5x_3 - 3x_4 &= 18 \\ 3x_3 + 5x_4 &= 18 \end{aligned}$$

Jawab:

Langkah pertama untuk menyelesaikannya, kita harus merubah persamaan tersebut kedalam bentuk matriks

$$\begin{bmatrix} 3 & 4 & 0 & 0 \\ 4 & 5 & 2 & 0 \\ 0 & 2 & 5 & 3 \\ 0 & 0 & 3 & 5 \end{bmatrix} x = \begin{bmatrix} 20 \\ 28 \\ 18 \\ 18 \end{bmatrix}$$

Untuk menyelesaikan persamaan tersebut menggunakan fungsi `tridiagmatrix()`, kita perlu membentuk vektor diagonal l, d, u, dan b.

```
l <- u <- c(4, 2, 3); d <- c(3, 5, 5, 5)
b <- c(20, 28, 18, 18)
```

Setelah terbentuk, vektor tersebut dapat langsung dimasukkan ke dalam fungsi `tridiagmatrix()`.

```
tridiagmatrix(L=l, D=d, U=u, b=b)
```

```
## [1] 4 2 1 3
```

Untuk menyelesaikannya menggunakan fungsi `gauss_jordan()`, kita perlu membentuk *augmented matrix*-nya terlebih dahulu.

```
m <- matrix(c(3,4,0,0,4,5,2,0,
              0,2,5,3,0,0,3,5,
              20,28,18,18), nrow=4)
gauss_jordan(m)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    4
## [2,]    0    1    0    0    2
## [3,]    0    0    1    0    1
## [4,]    0    0    0    1    3
```

6.3.4 Penyelesaian Sistem Persamaan Linier Menggunakan Fungsi `solve()`

`R` menyediakan fungsi bawaan `solve()` untuk menyelesaiakan sistem persamaan linier. Format fungsi `solve()` adalah sebagai berikut:

```r
solve(a,b)
```

Catatan:

- **a**: matriks koefisien atau matriks segiempat
- **b**: vektor konstanta

Berikut adalah contoh penerapan fungsi `solve()` pada sistem persamaan linier yang disajikan pada Contoh 6.2:

```r
# memecah matriks m menjadi matriks koefisien dan vektor konstanta
a <- matrix(c(2,3,1,1,2,-5,-1,-2,4),nrow=3)
b <- c(1,1,3)

solve(a,b)
```

```
## [1] 1 2 3
```

Jika kita hanya memasukkan matriks persegi, maka output yang akan dihasilkan adalah invers dari matriks yang kita masukkan.

```r
solve(a)
```

```
##      [,1] [,2]       [,3]
## [1,]    2   -1  7.401e-17
## [2,]   14   -9 -1.000e+00
## [3,]   17  -11 -1.000e+00
```

Jika kita mengalikan invers dengan matriks semula, maka akan dihasilkan output berupa matriks identitas.

```r
a%*%solve(a)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

6.3.5 Penyelesaian Sistem Persamaan Linier Menggunakan Fungsi 'Solve.tridiag()'

Penyelesaian matriks tridiagonal selain menggunakan fungsi `solve()`, juga dapat menggunakan fungsi `Solve.tridiag()` dari Paket `limSolve`. Untuk menginstall dan mengaktifkan Paket tersebut, jalankan sintaks berikut:

```
install.packages("limSolve")
```

```
library(limSolve)
```

Fungsi `Solve.tridiag()` memiliki format sebagai berikut:

```
Solve.tridiag ( diam1, dia, diap1, B=rep(0,times=length(dia)))
```

> **Catatan**:
>
> - **diam1**: vektor bukan nol di bawah diagonal matriks
> - **dia**: vektor bukan nol pada diagonal matriks
> - **diap1**: vektor bukan nol di atas diagonal matriks
> - **B**: vektor konstanta

Untuk memahami penerapannya, kita akan menggunakan kembali matriks yang ada pada Contoh 6.5.

```
l <- u <- c(4, 2, 3); d <- c(3, 5, 5, 5)
b <- c(20, 28, 18, 18)
Solve.tridiag(diam1=l, dia=d, diap1=u, B=b)
```

```
##      [,1]
## [1,]    4
## [2,]    2
## [3,]    1
## [4,]    3
```

6.4 Dekomposisi Matriks

Seringkali kita diminta untuk memperoleh nilai penyelesaian suatu persamaan linier $Ax = B$, dimana nilai vektor B yang selalu berubah-ubah. Penggunaan metode eliminasi Gauss mengharuskan untuk menyelesaikan sistem persamaan linier $Ax = B$ secara terpisah untuk setiap perubahan vektor B. Untuk menghindari pekerjaan eliminasi yang selalu berulang-ulang, faktorisasi menjadi suatu hal yang dapat dilakukan untuk mempersingkat prosesnya. Faktorisasi atau dekomposisi matriks merupakan suatu algoritma untuk memecah matriks A, hasil pemecahan ini selanjutnya digunakan untuk memperoleh penyelesaian sistem persamaan linier melalui perkalian antara vektor B dan hasil faktorisasi matriks A.

6.4.1 Dekomposisi LU

Misalkan kita memiliki persamaan linier seperti yang ditunjukkan oleh Persamaan (6.12). Pada metode dekomposisi LU, matriks A difaktorkan menjadi matriks L dan matriks U, dimana ukuran kedua matriks tersebut harus sama dengan ukuran matriks A atau dapat kita tuliskan bahwa hasil perkalian kedua matriks tersebut akan menghasilkan matriks A.

$$A = LU \tag{6.22}$$

Sehingga Persamaan (6.12) akan menjadi Persamaan (6.23).

$$LUx = b \tag{6.23}$$

Langkah penyelesaian sistem persamaan linier, diawali dengan menghadirkan vektor t yang ditunjukkan pada Persamaan (6.24).

$$Ux = t \tag{6.24}$$

Langkah pada Persamaan (6.24) tidak dimaksudkan untuk menghitung vektor t, melainkan untuk menghitung vektor x. Vektor t diperoleh dengan menggunakan Persamaan (6.25).

$$Lx = t \tag{6.25}$$

Kita dapat menyelesaikan sistem persamaan yang ditunjukkan pada Persamaan (6.24) dan Persamaan (6.25) menggunakan berbagai algoritma penyelesaian yang telah dibahas sebelumnya. Namun, karena matriks L merupakan matriks segitiga bawah dengan nilai nol berada pada bagian atas diagonal utama, penyelesaian t mengambil langkah yang lebih sedikit. Kondisi ini sama dengan kondisi penyelesaian matriks tridiagonal, dimana kita memanfaatkan sejumlah jalan pintas penyelesaiaannya guna mempercepat komputasi. Matriks segitia bawah L akan berupa matriks persegi dengan ukuran m, di mana m merupakan jumlah baris matriks A. Persamaan (6.25) dalam bentuk matriks akan terlihat seperti Persamaan (6.26).

$$Lt = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{m,1} & l_{m,2} & l_{m,3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \cdots \\ t_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.26}$$

Berdasarkan Persamaan (6.26), diketahui nilai $t_1 = b_1$. Nilai ini selanjutnya dapat digunakan untuk melakukan proses substitusi guna memperoleh seluruh nilai vektor t. Proses ini disebut sebagai *foward substitution*. Proses substitusi dapat dituliskan menggunakan Persamaan (6.27).

$$t_i = b_i - \sum_{j=1}^{i-1} l_{i,j} t_i \tag{6.27}$$

Seteleh nilai vektor t dihitung, kita dapat menghitung nilai x pada Persamaan (6.28).

$$Ux = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & u_{3,3} & \cdots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \cdots \\ t_n \end{bmatrix} \tag{6.28}$$

Jika diperhatikan, kita dapat mengetahui mengetahui nilai $x_n = \frac{t_n}{u_{m,n}}$. Nilai tersebut selanjutnya dapat digunakan untuk melakukan proses susbtitusi pada nilai lainnya. Proses substitusi ini disebut sebagai *backward substitution*. Proses dekomposisi atau faktorisasi LU digambarkan pada Gambar 6.1.

Dekomposisi LU didasarkan pada operasi baris elementer. Pertama, kita perlu menemukan matriks segitiga atas yang sesuai dengan matriks A. Solusi untuk melakukan dekomposisi bisa jadi tak terhingga, namun solusi yang paling sederhana adalah mengubah matriks A menjadi matriks *row echelon form*. Kedua, L harus menjadi matriks segitiga bawah yang mereduksi ke-l dengan mengikuti operasi baris yang sama yag menghasilkan U. Kita dapat menggunakan algoritma Doolittle untuk menghasilkan L, di mana nilai setiap entri dalam matriks segitiga bawah merupakan pengali yang digunakan untuk menghilangkan entri yang sesuai untuk setiap proses *row replacement*.

Pada praktiknya, proses eliminasi Gauss untuk memperoleh matriks U kadang menghasilkan nol di kolom pivotnya. Kondisi tersebut mengharuskan kita untuk melakukan proses *row swapping* atau pertukaran baris (biasanya dengan baris bawahnya) untuk pivot bukan nol. Jika proses tersebut berhasil dilakukan bisa jadi matriks A mungkin setara dengan matriks LU, tetapi tidak sama dalam hal

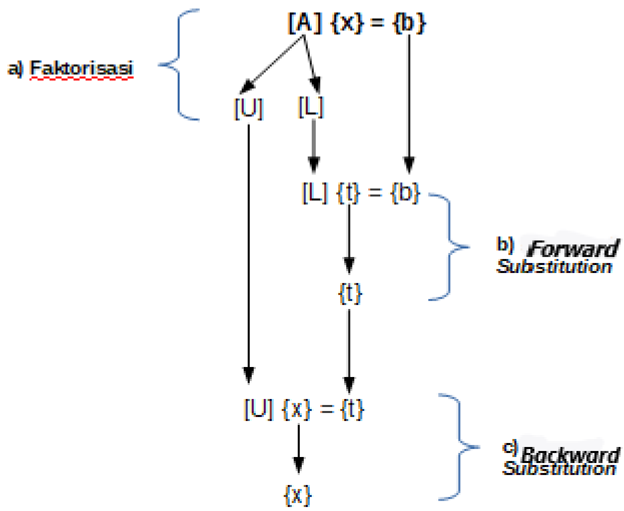

Figure 6.1: Tahapan dekomposisi LU.

urutan nilai pada tiap barisnya. Agar kita dapat memperoleh hasil yang sama (matriks A sama dengan matriks LU), diperlukan matriks ketiga, P. Matriks ini merupakan matriks identitas dengan ukuran sama dengan matriks A. Jika pertukaran baris dilakukan selama proses pembentukan matriks U, maka pertukaran baris yang sama juga akan diimplemenntasikan pada matriks P. oleh karena itu, dalam praktiknya matriks $A = PLU$ dan perkalian dengan matriks P berfungsi untuk mengembalikan urutan baris.

Contoh 6.6. Selesaikan sistem persamaan linier berikut menggunakan faktorisasi LU

$$\begin{gathered} x_1 + x_2 + 3x_4 = 4 \\ 2x_1 + x_2 - x_3 + x_4 = 1 \\ 3x_1 - x_2 - x_3 + 2x_4 = -3 \\ -x_1 - 2x_2 + 3x_3 - x_4 = 4 \end{gathered}$$

Jawab:

Nayatakan sistem persamaan tersebut ke dalam bentuk matriks $Ax = b$.

$$Ux = \begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix}$$

Lakukan operasi baris elementer pada matriks A untuk memperoleh matriks U. Urutan operasi baris elementer yang dilakukan adalah sebagai berikut:

- $(B_2 - 2B_1) \to B_2 \to l_{2,1} = 2$,
- $(B_3 - 3B_1) \to B_3 \to l_{3,1} = 3$,
- $(B_4 + B_1) \to B_4 \to l_{4,1} = -1$,
- $(B_3 - 4B_2) \to B_3 \to l_{3,2} = 4$,
- $(B_4 + 3B_2) \to B_4 \to l_{4,2} = -3$,
- $l_{4,3} = 0$

Simpan pengali tiap tahapan pada masing-masing elemen matriks L. Hasil operasi tersebut akan menghasilkan matriks triangular U.

$$U = \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix}$$

Untuk matriks L sebagai berikut:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix}$$

Karena pada proses operasi baris elementer tidak terdapat operasi pertukaran baris, maka matriks P tidak mengalami perubahan:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Lakukan operasi *forward substitution* menggunakan Persamaan (6.26).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix}$$

Berdasarkan hasil perhitungan diperoleh nilai vektor t.

$$t_1 = 4, t_2 = 1, t_3 = -3, t_4 = 4$$

Operasi terakhir yang perlu dilakukan untuk memperoleh nilai x adalah dengan melakukan *backward substitution* menggunakan nilai vektor t yang telah dihitung.

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 13 \\ -13 \end{bmatrix}$$

Berdasarkan hasil perhitungan diperoleh nilai x sebagai berikut:

$$x_1 = -1, x_2 = 2, x_3 = 0, x_4 = 1$$

Algoritma Dekomposisi LU

1. Masukkan matriks A, dan vektor B beserta ukurannya n
2. Lakukan langkah poin ke-4 s/d poin 5 untuk meperoleh matriks U.
3. Untuk baris ke-i di mana $i = 1$ s/d n, perhatikan apakah nilai $a_{i,j}$ sama dengan nol.

- **Bila iya**, lakukan *row swapping* antara baris ke-i dan baris ke-$i + k \leq n$, dimana $a_{i+k,j}$ tidak sama dengan nol. Bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian.
- **Bila tidak**, lanjutkan.

5. Untuk baris ke-j, dimana $j = i + 1$ s/d n, lakukan operasi baris elementer:

- Hitung $c = \frac{a_{j,i}}{a_{i,i}}$
- untuk kolom k, dimana $k = 1$ s/d $n + 1$, hitung $a_{j,k} = a_{j,k} - c_i.a_{i,k}$

6. Lakukan langkah poin ke-7 s/d poin 9 untuk memperoleh matriks L
7. Untuk diagonal matriks L isikan dengan nilai 1 dan elemen di atas diagonal dengan nilai nol.
8. Untuk elemen di bawah diagonal isikan dengan faktor pengali operasi baris elementer matriks U.
9. Lakukan proses *forward substitution* menggunakan Persamaan (6.27) untuk memperoleh nilai vektor t.

10. Lakukan *backward substituion* menggunakan Persamaan (6.16).

Berdasarkan algoritma tersebut, kita dapat menyusun algoritma faktorisasi LU menggunakan `R`. Berikut adalah sintaks yang digunakan:

```
lu_solve <- function(a, b=NULL){
    m <- nrow(a)
    n <- ncol(a)
    piv <- 1

# membentuk matriks identitas P dan L
    P <- L <- diag(n)

# cek elemen diagonal utama apakah bernilai nol
    for(row_curr in 1:m){
        if(piv <= n){
            i <- row_curr
            while(a[i, piv] == 0 && i < m){
                i <- i + 1
                if(i > m){
                    i <- row_curr
                    piv <- piv + 1
                    if(piv > n)
                        return(list(P = P, L = L, U = a))
                }
            }

# jika elemen diagonal utama bernilai nol,lakukan row swapping
            if(i != row_curr){
                a <- swap_row(a, i, row_curr)
                P <- swap_row(P, i, row_curr)
            }

  # pembentukan matriks L dan U
            for(j in row_curr:m)
                if(j != row_curr){
                    k <- a[j, piv]/a[row_curr, piv]

  # matriks U
                    a <- replace_row(a, row_curr, j, -k)

  # pengisian elemen matriks L
                    L[j, piv] <- k
```

```
                }
            piv <- piv + 1
        }
    }

# penyelesaian persamaan linier
    if(is.null(b)){
      return(list(P = P, L = L, U = a))
    }else{

      # forward substitution
      t <- forwardsolve(L, b)

      # backward substitution
      x <- backsolve(a, t)
      return(list(P = P, L = L, U = a, result=x))
     }
}
```

Kita dapat menyelesaikan sistem persamaan linier pada Contoh 6.6 menggunakan fungsi yang telah kita buat.

```
# membuat matriks a dan vektor b
a <- matrix(c(1,2,3,-1,1,1,-1,2,
              0,-1,-1,3,3,1,2,-1),
            nrow=4)
b <- c(4,1,-3,4)

# penyelesaian
decomp<-lu_solve(a,b)
```

Untuk membentuk kembali matriks A, kita dapat mengalikan matriks L, U, dan P.

```
decomp$L%*%decomp$U%*%decomp$P
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    0    3
## [2,]    2    1   -1    1
## [3,]    3   -1   -1    2
## [4,]   -1    2    3   -1
```

Contoh 6.7. Lakukan dekomposisi LU pada matriks berikut dan lakukan pengecekan apakah perkalian hasil dekomposisi matriks akan menghasilkan matriks semula!

$$\begin{bmatrix} 0 & 1 & -1 \\ 1 & 5 & 9 \\ 7 & -1 & -5 \end{bmatrix}$$

Jawab:

Lakukan proses dekomposisi menggunakan fungsi `lu_solve()`.

```
# membentuk matriks a
(A <- matrix(c(0, 1, 7, 1, 5, -1, -2, 9, -5), 3))
```

```
##      [,1] [,2] [,3]
## [1,]    0    1   -2
## [2,]    1    5    9
## [3,]    7   -1   -5
```

```
# dekomposisi lu
decomp<-lu_solve(A)
```

Lakukan pengecekan apakah matriks hasil dekomposisi akan menghasilkan matriks A.

```
decomp$P %*% decomp$L %*% decomp$U
```

```
##      [,1] [,2] [,3]
## [1,]    0    1   -2
## [2,]    1    5    9
## [3,]    7   -1   -5
```

Fungsi `lu()` pada Paket `Matrix` dapat digunakan untuk melakukan dekomposisi LU. Untuk meggunakan fungsi tersebut, kita harus menginstall dan mengaktifkan Paket `Matrix`.

```
install.packages("Matrix")
```

```
library(Matrix)
```

Untuk dapat menggunakannya kita hanya perlu menginputkan matriks kedalam fungsi tersebut. Berikut adalah contoh penerapannya:

```
# membuat matriks a
a <- Matrix::Matrix(round(rnorm(9),2), nrow=3)

# dekomposisi
lum <- Matrix::lu(a)
lum
```

```
## 'MatrixFactorization' of Formal class 'denseLU' [package "Matrix"] with 4 slots
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:9] -0.95 0.705 0.589 0.53 -0.604 ...
##   ..@ perm    : int [1:3] 2 3 3
##   ..@ Dim     : int [1:2] 3 3
```

Untuk menampilkan hasil dekomposisi, jalankan fungsi `expand()`.

```
decomp <- Matrix::expand(lum)
decomp
```

```
## $L
## 3 x 3 Matrix of class "dtrMatrix" (unitriangular)
##      [,1]    [,2]    [,3]
## [1,]  1.0000       .       .
## [2,]  0.7053  1.0000       .
## [3,]  0.5895 -0.2279  1.0000
##
## $U
## 3 x 3 Matrix of class "dtrMatrix"
##      [,1]    [,2]    [,3]
## [1,] -0.9500  0.5300  1.7600
## [2,]       . -0.6038 -0.7513
## [3,]       .       .  0.1913
##
## $P
## 3 x 3 sparse Matrix of class "pMatrix"
##
## [1,] . . |
## [2,] | . .
## [3,] . | .
```

6.4.2 Dekomposisi Cholesky

Dekomposisi Cholesky memberikan faktorisasi matriks alternatif sehingga $A = LL^T$, di mana L^T merupakan transpose konjugat dari matriks L. Dalam kasus ini, penulis hanya bekerja dengan matriks rill dengan nilai rill dan bagian imajiner nol. Jadi untuk tujuan *sub-chapter* ini, matriks L^T hanyalah transpose dari matriks L.

Seperti dekomposisi LU, dekomposisi Cholesky dapat digunakan untuk menyelesaikan sistem persamaan linier. Kelebihannya, Menemukan dekomposisi Cholesky jauh lebih cepat daripada dekomposisi LU. Namun, dekomposisi ini hanya terbatas pada matriks tertentu saja. Dekomposisi Cholesky hanya dapat digunakan pada matriks definit positif dan simetris. Matriks simeteris merupakan matriks yang nilai di atas dan di bawah diagonalnya simetris atau sama; secara matematis, untuk semua i dan j pada matriks A, $a_{i;j} = a_{j;i}$. Definit positif berarti bahwa setiap entri pivot (nilai elemen diagonal utama) selelu bernilai positif. Selain itu, untuk matriks definit positif, hubungan $xAx > 0$ untuk semua vektor, x.

Karena L^* transpose dari matriks L, maka $l^T_{i,j} = l_{j,i}$ untuk semua nilai i dan j. Tanpa kendala (*constraint*) ini, dekomposisi Cholesky akan mirip dekomposisi LU. Tetapi dengan kendala ini, nilai elemen matriks L dan L^T harus dipilih dengan cermat sehingga hubungan $A = LL^T$ berlaku. Bentuk dekomposisi Cholesky disajikan pada Persamaan (6.29).

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,m} \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{m,1} & l_{m,2} & l_{m,3} & \cdots & l_{m,m} \end{bmatrix} \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & \cdots & l_{1,m} \\ 0 & l_{2,2} & l_{2,3} & \cdots & l_{2,m} \\ 0 & 0 & l_{3,3} & \cdots & l_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & l_{m,m} \end{bmatrix} \tag{6.29}$$

Untuk setiap elemen matriks A memiliki hubungan yang dituliskan pada Persamaan (6.30).

$$a_{i,j} = \sum_{k=1}^{m} L_{i,k} L_{k,j} \tag{6.30}$$

Berdasarkan Persamaan (6.29), sejumlah nilai elemen $L_{i,k}$ dan $L_{k,j}$ adalah nol. Nilai tiap elemen diagonal utama yang tidak bernilai nol dihitung menggunakan Persamaan (6.31).

$$l_{i,i} = \sqrt{\left(a_{i,i} - \sum_{k=1}^{i-1} l^2_{i,k} \right)} \tag{6.31}$$

Elemen diagonal dihitung menggunakan Persamaan (6.32)

$$l_{i,j} = \frac{1}{l_{i,i}} \left(a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} l_{j,k} \right) \tag{6.32}$$

Algoritma Dekomposisi Cholesky

1. Masukkan matriks A, dan vektor B beserta ukurannya n.
2. Untuk elemen matriks L, hitung menggunakan Persamaan (6.32).
3. Untuk nilai diagonal utama matriks L, hitung menggunakan Persamaan (6.31).
4. Untuk memperoleh matriks L^T, lakukan transpose pada matriks L.
5. Untuk memperoleh nilai x,

- Hitung vektor t menggunakan Persamaan (6.25).
- Hitung vektor x menggunakan Persamaan (6.24), dimana matriks $U = L^T$.

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi pada `R` untuk melakukan dekomposisi Cholesky. Fungsi tersebut disajikan pada sintaks berikut:

```
cholesky_solve <- function(a, b=NULL){
    m <- nrow(a)

# membentuk matriks L dengan elemen nol
    L = diag(0,m)

# Perhitungan elemen matriks L
    for(i in 1:m){
        for(k in 1:i){
            p_sum <- 0
            for(j in 1:k)
                p_sum <- p_sum + L[j,i]*L[j,k]

# Pehitungan elemen diagonal utama
            if(i==k)
                L[k,i]<-sqrt(a[i,i]-p_sum)
            else
                L[k,i]<-(a[k,i]-p_sum)/L[k,k]
        }
```

```
    }

# Perhitungan elemn matriks L*
    tL <- t(L)

# penyelesaian persamaan linier
    if(is.null(b)){
      return(list(L = L, tL = tL, a = a))
    }else{

      # forward substitution
      t <- forwardsolve(L, b)

      # backward substitution
      x <- backsolve(tL, t)
      return(list(L = L, tL = tL, a = a, result=x))
     }
}
```

Contoh 6.8. Dengan menggunakan fungsi `cholesky_solve()`, lakukan dekomposisi pada matriks berikut! Lakukan pengecekan pada hasil dekomposisi apakah hasil kali matriks dekomposisi akan menghasilkan matriks semula!

$$\begin{bmatrix} 9 & -3 & 6 \\ -3 & 17 & -10 \\ 6 & -10 & 12 \end{bmatrix}$$

Jawab:

Dekomposisi Cholesky menggunakan fungsi `cholesky_solve()`, disajikan pada sintaks berikut:

```
a <- matrix(c(9,-3,6,-3,17,-10,6,-10,12),3)

# dekomposisi Cholesky
(decomp<-cholesky_solve(a))
```

```
## $L
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    0    4   -2
## [3,]    0    0    2
##
```

```
## $tL
##      [,1] [,2] [,3]
## [1,]    3    0    0
## [2,]   -1    4    0
## [3,]    2   -2    2
##
## $a
##      [,1] [,2] [,3]
## [1,]    9   -3    6
## [2,]   -3   17  -10
## [3,]    6  -10   12
```

```
# mengecek hasil dekomposisi
decomp$tL %*% decomp$L
```

```
##      [,1] [,2] [,3]
## [1,]    9   -3    6
## [2,]   -3   17  -10
## [3,]    6  -10   12
```

Fungsi lain yang dapat digunakan untuk melakukan dekomposisi Cholesky adalah menggunakan fungsi `chol()` pada Paket `Matrix`. Pada fungsi tersebut, kita hanya perlu menginputkan objek matrik kedalamnya. Berikut adalah contoh penerapan fungsi tersebut menggunakan matriks pada Contoh 6.8.

```
chol(a)
```

```
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    0    4   -2
## [3,]    0    0    2
```

> **Penting!!!**
>
> Fungsi `chol()` hanya menampilkan matriks L^T. Untuk menampilkan matriks L, kita perlu melakukan transpose

6.4.3 Dekomposisi Lainnya

Terdapat beberapa algoritma lain yang telah dikembangkan untuk melakukan dekomposisi matriks. Pada buku ini hanya akan dijelaskan secara singkat terkait fungsi yang digunakan dalam melakukan dekomposisi matriks. Algoritma yang akan dijelaskan pada *sub-chapter* ini antara lain: QR, *singular value decomposition* (SVD), dan dekomposisi eigen. Untuk algoritma lainnya, pembaca dapat membaca buku terkait atau mengecek dokumentasinya pada Paket `base`.

6.4.3.1 Dekomposisi QR

Dekomposisi QR merupakan dekomposisi yang penting dalam menyelesaikan sistem persamaan linier. Dekomposisi ini juga berperan penting untuk menghitung koefisien regresi dan pengaplikasian algoritma Newton-Raphson.

Untuk memperoleh informasi terkait dekomposisi ini, pembaca dapat mengetikkan sintaks berikut pada `R`:

```
?qr
```

Berikut merupakan contoh penerapan fungsi `qr()` untuk menyelesaikan sistem persamaan linier:

```
# membuat matriks A dan B
set.seed(123)
A <- matrix((1:12)+rnorm(12), nrow=4)
b <- 2:5

# dekomposisi matriks A
qr(A)
```

```
## $qr
##         [,1]     [,2]     [,3]
## [1,] -6.3778 -12.1257 -19.8501
## [2,]  0.2775  -6.3105  -7.9392
## [3,]  0.7148  -0.6461   2.3512
## [4,]  0.6382  -0.5654   0.2767
##
## $rank
## [1] 3
##
## $qraux
## [1] 1.069 1.513 1.961
##
## $pivot
## [1] 1 2 3
##
## attr(,"class")
## [1] "qr"
```

```
# memperoleh penyelesaian SPL
qr.solve(A,b)
```

```
## [1]  0.3046 -0.1111  0.3237
```

6.4.3.2 *Singular Value Decomposition*

Singular value decomposition (SVD) merupakan algoritma faktorisasi matriks yang mendekomposisi matriks segiempat menjadi matriks UDV_H, dimana D merupakan mmatriks diagonal non negatif, U dan V merupakan matriks *unitary*, dan V_H merupakan matriks tanspose konjugat dari matriks V. Algoritma ini banyak digunakan dalam analisis *principal component*.

Pada `R`, SVD dapat dilakukan menggunakan fungsi `svd()` dari Paket `base`. Berikut adalah sintaks untuk memperoleh informasi terkait fungsi tersebut:

```
?svd
```

Berikut adalah contoh penerapan fungsi `svd()`:

```
# dekomposisi matriks A
svd(A)
```

```
## $d
## [1] 26.094  2.727  1.330
##
## $u
##         [,1]    [,2]    [,3]
## [1,] -0.3685 -0.5661  0.6651
## [2,] -0.4707 -0.5703 -0.6113
## [3,] -0.5740  0.4324 -0.2590
## [4,] -0.5596  0.4089  0.3419
##
## $v
##         [,1]     [,2]    [,3]
## [1,] -0.2257  0.87169 -0.4350
## [2,] -0.5202 -0.48536 -0.7028
## [3,] -0.8237  0.06764  0.5630
```

6.4.3.3 Dekomposisi Eigen

Proses umum yang digunakan untuk menemukan nilai eigen dan vektor eigen suatu matriks segiempat dapat dilihat sebagai proses dari dekomposisi eigen. Proses ini akan mendekomposisi matriks menjadi VDV^{-1}, dimana D merupakan matriks diagonal yang terbentuk dari nilai eigen, dan V merupakan vektor eigen. Proses dekomposisi ini akan berguna bagi pembaca yang ingin mempelajari *principal component analysis*.

Fungsi `eigen()` pada Paket `base` dapat digunakan untuk melakukan dekomposisi eigen. Untuk mempelajari lebih jauh terkait fungsi ini, pambaca dapat menjalankan sintaks berikut:

```
?eigen
```

Berikut adalah contoh sintaks untuk melakukan dekomposisi eigen:

```
A <- matrix(c(2,-1,0,-1,2,-1,0,-1,2), nrow=3)

# dekomposisi matriks A
eigen(A)
```

```
## eigen() decomposition
## $values
## [1] 3.4142 2.0000 0.5858
##
## $vectors
##         [,1]       [,2]   [,3]
## [1,] -0.5000 -7.071e-01 0.5000
## [2,]  0.7071  1.099e-15 0.7071
## [3,] -0.5000  7.071e-01 0.5000
```

6.5 Metode Iterasi

Pada Chapter 6.5 kita akan membahas penyelesaian persamaan linier dengan menggunakan metode iterasi. Terdapat dua metode iterasi yang akan dibahas yaitu iterasi Jacobi dan Gauss-Seidel.

Metode iterasi dimulai dengan estimasi nilai akhir. Setelah menerapkan beberapa perlakuan pada nilai estimasi, hasil perlakuan selanjutnya menjadi nilai estimasi untuk iterasi berikutnya. Proses tersebut akan berlangsung secara terus-menerus hingga ambang batas dipenuhi. Nilai ambang batas dapat berupa jumlah iterasi maksimum atau selisih antara nilai estimasi baru dan estimasi semula lebih kecil dari suatu nilai toleransi yang ditetapkan.

Jumlah kuadrat merupakan metode yang sering digunakan untuk mengecek apakah selisih nilai estimasi baru terhadap estimasi lama lebih kecil dari nilai toleransi yang ditetapkan. Persamaan (6.33) menampilkan hubungan antara jumlah kuadrat dan nilai toleransi pada proses iterasi.

$$\sqrt{\sum_{i=1}^{n}\left(x_i^{n+1}-x_i^{n}\right)^2} < t_0 \tag{6.33}$$

dimana x^n merupakan iterasi ke-n dari algoritma dan t_0 merupakan nilai toleransi maksimum yang diterima.

6.5.1 Iterasi Jacobi

Untuk menyelesaikan matriks menggunakan metode iterasi, kita dapat mulai dengan premis terdapat matriks A dan vektor x dan b, sehingga $Ax = b$. Dengan menggunakan metode Jacobi, pertama-tama kita dapat amati bahwa terdapat matriks R dan D yang memiliki hubungan $A = R + D$. Berdasarkan kedua hubungan tersebut, dapat diturunkan operasi matriks melalui persamaan berikut:

$$Ax = b \tag{6.34}$$

$$Rx + Dx = b \tag{6.35}$$

$$Dx = b - Rx \tag{6.36}$$

$$x = D^{-1}\left(b - Rx\right) \tag{6.37}$$

Persamaan (6.37) merupakan persamaan yang dapat kita gunakan untuk memperoleh nilai x. Jika kita menulis kembali persamaan tersebut, maka kita akan memperoleh persamaan yang digunakan sebagai acuan iterasi Jacobi.

$$x^{n+1} = D^{-1}\left(b - Rx^{n}\right) \tag{6.38}$$

dimana D merupakan matriks diagonal dengan nilai elemen diagonal berupa diagonal utama matriks A. Invers dari matriks D secara sederhana sebagai matriks diagonal sama dengan satu dibagi dengan elemen diagonal utama matriks A. Matriks R identik dengan matriks A. Namun, diagonal utamanya bernilai nol. Suatu iterasi dikatakan konvergen jika jumlah kuadrat dari vektor $x^{(n+1)}$ dan vektor $x^{(n)}$ semakin mengecil.

Suatu persamaan linier yang hendak diselesaikan dengan menggunakan metode iterasi Jacobi harus memenuhi syarat nilai elemen diagonal utama matriks harus lebih dominan. Maksudnya adalah nilai absolut diagonal utama matriks harus lebih besar dari jumlah nilai absolut elemen matriks lainnya pada satu kolom.

Contoh 6.9. Selesaikan sistem persamaan berikut menggunakan iterasi Jacobi!

$$\begin{bmatrix} 5 & 2 & 3 \\ 2 & 7 & 4 \\ 1 & 3 & 8 \end{bmatrix} x = \begin{bmatrix} 40 \\ 39 \\ 55 \end{bmatrix}$$

Jawab:

Berdasarkan matriks A (matriks koefisien), kita dapat memastikan bahwa matriks tersebut memiliki nilai dominan pada elemen diagonal utama. Sebagai contoh:

$$|5| > |2| + |1| \quad (kolom\ 1)$$

$$|7| > |2| + |3| \quad (kolom\ 2)$$

Untuk mempermudah proses iterasi, kita akan menggunakan bantuan `R` untuk melakukan komputasi. Langkah pertama yang perlu dilakukan adalah menyiapkan matriks A, vektor b, dan vektor x (nilai taksiran awal).

```
(A <- matrix(c(5,2,1,2,7,3,3,4,8), 3))
```

```
##      [,1] [,2] [,3]
## [1,]    5    2    3
## [2,]    2    7    4
## [3,]    1    3    8
```

```
(b <- c(40,39,55))
```

```
## [1] 40 39 55
```

```
(x <- rep(0,3))
```

```
## [1] 0 0 0
```

Langkah selanjutnya adalah memperoleh invers matriks D.

```
(Dinv <- diag(1/diag(A)))
```

```
##      [,1]   [,2]  [,3]
## [1,]  0.2 0.0000 0.000
## [2,]  0.0 0.1429 0.000
## [3,]  0.0 0.0000 0.125
```

Persiapan terakhir sebelum iterasi dilakukan adalah menyiapkan matriks R.

```
(R<-A-diag(diag(A)))
```

```
##      [,1] [,2] [,3]
## [1,]    0    2    3
## [2,]    2    0    4
## [3,]    1    3    0
```

Iterasi selanjutnya dilakukan menggunakan Persamaan (6.38).

iterasi 1

```
(x1 <- Dinv %*% (b-R%*%x))
```

```
##       [,1]
## [1,] 8.000
## [2,] 5.571
## [3,] 6.875
```

iterasi 2

```
(x2 <- Dinv %*% (b-R%*%x1))
```

```
##         [,1]
## [1,]  1.6464
## [2,] -0.6429
## [3,]  3.7857
```

iterasi 3

```
(x3 <- Dinv %*% (b-R%*%x2))
```

```
##       [,1]
## [1,] 5.986
## [2,] 2.938
## [3,] 6.910
```

Selama proses iterasi,jumlah akar jumlah kuadrat dihitung. Sebagai contoh berikut disajikan akar jumlah kuadrat pada iterasi ke-3:

```
sqrt(sum(x3-x2)^2)
```

```
## [1] 11.04
```

Selama proses iterasi nilai tersebut terus mengecil. Iterasi dihantikan jika nilai akar jumlah kuadrat tersebut lebih kecil dari nilai toleransi. Pada contoh ini digunakan nilai toleransi 10^{-7}.

Proses iterasi berlangsung sampai dengan iterasi ke-62 dengan nilai x akhir sebagai berikut:

$$x = \begin{bmatrix} 4 \\ 1 \\ 6 \end{bmatrix}$$

Algoritma Iterasi Jacobi

1. Masukkan matriks A, dan vektor B beserta ukurannya n.
2. Hitung invers matriks D, dimana nilai invernya merupakan matriks diagonal dari satu per diagonal utama matriks A.
3. Hitung matriks R, dimana R merupakan selisih matriks A dikurangi dengan matriks diagonal dengan entri dari diagonal utama matriks A.
4. Tetapkan vektor x estimasi.
5. Tetapkan nilai toleransi maksimum yang dapat diterima.
6. Lakukan iterasi menggunakan Persamaan (6.38).
7. Hitung akar jumlah kuadrat dari vektor x^{n+1} dan vektor x^n.
8. Jadikan nilai x^{n+1} sebagai nilai taksiran x untuk iterasi berikutnya.
9. Hentikan proses iterasi jika telah memenuhi syarat yang ditampilkan pada Persamaan (6.33).

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi sebuah fungsi untuk melakukan iterasi Jacobi. Berikut sintaks yang digunakan:

```
jacobi <- function(a, b, tol=1e-7, maxiter=100){
  n <- length(b)
  iter <- 0

  Dinv <- diag(1/diag(a))
  R <- a-diag(diag(a))
  x <- rep(0,n)
  x_new <- rep(tol, n)

  while(sqrt(sum(x_new-x)^2)>tol){
          if(iter>maxiter){
```

```
              warning("iterasi maksimum tercapai")
              break
            }
            x <- x_new
            x_new <- Dinv %*% (b - R %*% x)
            iter <- iter+1
  }
  return(list(X = x_new, iter=iter))

}
```

Berikut adalah penerpan fungsi `jacobi()` tersebut:

```
jacobi(A,b)
```

```
## $X
##      [,1]
## [1,]    4
## [2,]    1
## [3,]    6
##
## $iter
## [1] 62
```

Contoh 6.10. Selesaikan sistem persamaan berikut menggunakan fungsi `jacobi()`

$$\begin{bmatrix} 27 & 6 & -1 \\ 6 & 15 & 2 \\ 1 & 1 & 54 \end{bmatrix} x = \begin{bmatrix} 85 \\ 72 \\ 110 \end{bmatrix}$$

Jawab:

Matriks A (matriks koefisien) berdasarkan sistem persamaan linier tersebut telah memenuhi syarat dari algoritma Jacobi (nilai diagonal utama dominan dibanding nilai lainnya pada satu kolom). Penyelesaian sistem persamaan tersebut, sebagai berikut:

```
A <- matrix(c(27,6,1,6,15,1,-1,2,54), 3)
b <- c(85,72,110)

jacobi(A,b)
```

```
## $X
##       [,1]
## [1,] 2.425
## [2,] 3.573
## [3,] 1.926
##
## $iter
## [1] 17
```

Nilai vektor x sesungguhnya dapat diperoleh menggunakan fungsi `solve()`.

```
solve(A,b)
```

```
## [1] 2.425 3.573 1.926
```

Berdasarkan hasil perhitungan, vektor x hasil iterasi memiliki nilai identik dengan nilai penyelesaian yang sebenarnya.

Perlu diperhatikan dalam penggunaan fungsi `jacobi()` syarat utama matriks haruslah terpenuhi, seperti: nilai diagonal matriks A lebih besar dari nilai elemen lainnya pada satu kolom. Selain itu, nilai diagonal matriks D tidak boleh sama dengan nol agar inver matriks D dapat diperoleh. Jika syarat-syarat tersebut terpenuhi, maka metode Jacobi dapat diterapkan. Jika tidak terpenuhi, maka penyelesaian yang konvergen mungkin masih dapat diperoleh meskipun penulis tidak dapat menjamin hal tersebut dapat terjadi.

6.5.2 Iterasi Gauss-Seidel

Metode iterasi Gauss-Seidel melakukan dekomposisi pada matriks A menjadi matriks segitiga atas U dan matriks segitiga bawah L. Dekomposisi ini tidak sama dengan dekomposisi LU pada Chapter 6.4.1. Matriks U pada metode Gauss-Seidel merupakan elemen (entri) matriks A pada bagian atas diagonal utama, sedangkan matriks L merupakan elemen diagonal utama dan bagian bawah diagonal utama matriks A. Elemen selain yang penulis sebutkan pada kedua matriks tersebut akan bernilai nol. Persamaan iterasi Gauss-Seidel ditampilkan pada Persamaan (6.39).

$$x^{n+1} = L^{-1}\left(b - Ux^n\right) \tag{6.39}$$

Syarat agar suatu sistem persamaan linier dapat diselesaikan menggunakan metode Gauss-Seidel adalah matriks harus memiliki nilai diagonal utama yang dominan. Maksudnya, nilai absolut diagonal utama lebih besar dari jumlah nilai absolut elemen lainnya dalam satu kolom. Jika syarat ini tidak terpenuhi maka metode ini tidak akan memperoleh penyelesaian yang konvergen.

Contoh 6.11. Selesaikan sistem persamaan pada Contoh 6.10 menggunakan iterasi Gauss-Seidel!

Jawab:

Kita akan kembali menggunakan bantuan `R` untuk melakukan kalkulasi pada proses iterasi Gauss-Seidel. Kita telah melakukan pengecekan pada sistem persamaan linier pada contoh tersebut dan menghasilkan kesimpulan bahwa persamaan linier tersebut dapat diselesaikan dengan metode Gauss-Seidel. Langkah selanjutnya adalah membentuk matriks L dan U.

```
# membentuk matriks U dan L dari matriks A
(L <- U <- A)
```

```
##      [,1] [,2] [,3]
## [1,]   27    6   -1
## [2,]    6   15    2
## [3,]    1    1   54
```

```
# membentuk matriks L dari entri bagian bawah diagonal utama matriks A
L[upper.tri(A, diag=FALSE)]<-0
L
```

```
##      [,1] [,2] [,3]
## [1,]   27    0    0
## [2,]    6   15    0
## [3,]    1    1   54
```

```
# membentuk matriks U dari entri bagian atas diagonal utama matriks A
U[lower.tri(A, diag=TRUE)]<-0
U
```

```
##      [,1] [,2] [,3]
## [1,]    0    6   -1
## [2,]    0    0    2
## [3,]    0    0    0
```

Selanjutya lakukan invers terhadap matriks L menggunakn fungsi `solve()`.

```
(Linv <- solve(L))
```

```
##            [,1]      [,2]    [,3]
## [1,]  0.0370370  0.000000 0.00000
## [2,] -0.0148148  0.066667 0.00000
## [3,] -0.0004115 -0.001235 0.01852
```

Tetapkan nilai estimasi awal dan nilai toleransi yang dikehendaki. Nilai toleransi pada proses ini ditetapkan sebesar 10^-7.

```
# tebakan awal nilai x
(x <- rep(0, length(b)))
```

```
## [1] 0 0 0
```

Lakukan iterasi menggunakan Persamaan (6.39).

Iterasi 1

```
(x1 <- Linv %*% (b - U %*% x))
```

```
##       [,1]
## [1,] 3.148
## [2,] 3.541
## [3,] 1.913
```

```
# akar jumlah kuadrat
sqrt(sum(x1-x)^2)
```

```
## [1] 8.602
```

Iterasi 2

```
(x2 <- Linv %*% (b - U %*% x1))
```

```
##       [,1]
## [1,] 2.432
## [2,] 3.572
## [3,] 1.926
```

```
# akar jumlah kuadrat
sqrt(sum(x2-x1)^2)
```

```
## [1] 0.672
```

Iterasi terus dilakukan sampai dengan nilai akar jumlah kuadrat lebih kecil dari nilai toleransi. Setelah iterasi ke-7 diperoleh nilai vektor x sebesar:

$$x = \begin{bmatrix} 2,425476 \\ 3,573016 \\ 1,925954 \end{bmatrix}$$

Algoritma Iterasi Gauss-Seidel

1. Masukkan matriks A, dan vektor B beserta ukurannya n.
2. Lakukan dekomposisi LU, dimana matriks L merupakan matriks segitiga bawah dengan nilai entri diagonal utama matriks A dan bagian bawah diagonalnya dan matriks U merupakan matriks segitiga atas dengan entri berasal dari elemen atas diagonal utama matriks A. Isi elemen lain yang tidak disebut pada kedua matriks tersebut dengan nol.
3. Tetapkan vektor x estimasi.
4. Tetapkan nilai toleransi maksimum yang dapat diterima.
5. Lakukan iterasi menggunakan Persamaan (6.39).
6. Hitung akar jumlah kuadrat dari vektor x^{n+1} dan vektor x^n.
7. Jadikan nilai x^{n+1} sebagai nilai taksiran x untuk iterasi berikutnya.
8. Hentikan proses iterasi jika telah memenuhi syarat yang ditampilkan pada Persamaan (6.33).

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi sebuah fungsi untuk melakukan iterasi Gauss-Seidel. Berikut sintaks yang digunakan:

```
gauss_seidel <- function(a, b, tol=1e-7, maxiter=100){
  n <- length(b)
  iter <- 0


  L <- U <- a
  L[upper.tri(a, diag=FALSE)] <- 0
  U[lower.tri(a, diag=TRUE)] <- 0
  Linv <- solve(L)

  x <- rep(0,n)
  x_new <- rep(tol, n)

  while(sqrt(sum(x_new-x)^2)>tol){
            if(iter>maxiter){
              warning("iterasi maksimum tercapai")
              break
            }
            x <- x_new
            x_new <- Linv %*% (b - U %*% x)
            iter <- iter+1
```

```
  }
      return(list(X = x_new, iter=iter))
}
```

Contoh 6.12. Selesaikan sistem persamaan pada Contoh 6.10 menggunakan fungsi `gauss_seidel()`!

Jawab:

Penyelesaiansistem persamaan linier tersebut menggunakan fungsi `gauss_seidel()` disajikan pada sintaks berikut:

```
gauss_seidel(A,b)
```

```
## $X
##       [,1]
## [1,] 2.425
## [2,] 3.573
## [3,] 1.926
##
## $iter
## [1] 7
```

6.6 Studi Kasus

Aljabar linier banyak diaplikasikan baik dalam bidang *engineering*, fisika, sampai dengan statistika. Pada *sub-chapter* ini penulis akan menjelaskan penerapan aljabar linier pada metode kuadrat terkecil dan aliran massa dalam reaktor. Untuk penerapan lainnya pembaca dapat membaca buku lainnya terkait aljabar linier.

6.6.1 Metode Kuadrat Terkecil

Metode kuadrat terkecil merupakan salah satu aplikasi penerapan aljabar linier yang paling populer. Intuisi dibalik metode ini adalah bagaimana kita meminimalkan jarak antara sejumlah titik dengan garis regresi. Misalkan kita menggambarkan scatterplot antara dua buah variabel. Pola yang terbentuk dari plot tersebut adalah terjadi korelasi positif antara variabel pada sumbu x dan sumbu y. Kita ingin menggambarkan garis regresi terbaik yang dapat menangkap seluruh pola tersebut. Garis regresi terbaik terjadi ketika jumlah kuadrat jarak antara titik observasi dan garis regresi yang terbentuk seminimal mungkin.

Untuk lebih memahaminya kita akan melakukan latihan menggunakan dataset `trees` yang berisi data hasil pengukuran kayu dari pohon yang ditebang. Pada dataset ini terdapat 31 observasi dan 3 buah kolom. Keterangan dari ketiga buah kolom tersebut adalah sebagai berikut:

- `Girth`: diameter pohon dalam satuan *inch*.
- `Height`: tinggi pohon dalam satuan *feet*.
- `Volume`: volume kayu dalam satuan *cubic feet*.

Untuk mengecek 6 observasi pertama dan struktur data, jalankan sintaks berikut:

```
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
```

```
str(trees)
```

```
## 'data.frame':    31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

Scatterplot matriks sangat bagus untuk mengecek korelasi antar variabel dalam dataset tersebut. Berikut adalah sintaks untuk membuatnya:

Kita ingin membuat sebuah model linier untuk memprediksi `Volume` kayu berdasarkan variabel `Girth` dan `Heiht` atau volume sebagia fungsi dari variabel `Girth` dan `Heiht`. Kita dapat menuliskan relasi antara variabel volume sebagai fungsi dari variabel `Girth` dan `Heiht` menggunakan Persamaan (6.40).

$$Volume = \beta_{girth} Girth + \beta_{height} Height + \beta_0 \tag{6.40}$$

dimana β_0 merupakan intersep persamaan regresi linier dan nilai β lainnya merupakan koefisien dari variabel `Girth` dan `Heiht`. Variabel `Volume` disebut sebagai variabel respon, sedangkan variabel `Girth` dan `Heiht` disebut sebagai variabel prediktor.

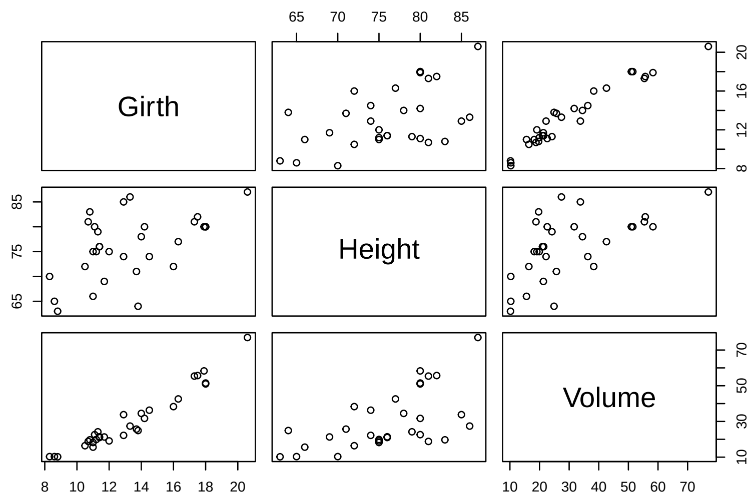

Figure 6.2: Scatterplot matriks dataset trees
(#fig:trees, LUfig)

Metode kuadrat terkecil berusaha memperoleh seluruh koefisien variabel dan intersep dari persamaan regresi linier. Berdasarkan yang telah penulis jelaskan garis regresi terbaik adalah garis yang memiliki nilai kuadrat terkecil jarak antara titik observasi dan garis regresi. Dasar dari metode kuadrat terkecil merupakan persamaan yang relatif sederhana yang ditunjukkan pada Persamaan (6.41).

$$A^T A = A^T b \tag{6.41}$$

dimana b merupakan vektor dari variabel respon (`Volume`) dan matrik A merupakan matriks variabel prediktor (variabel `Girth` dan `Heiht`).

Untuk menginputkan intercept kedalam persamaan linier kita perlu menmabhakan satu kolom di awal matriks A yang berisi nilai 1. Berikut adalah sintaks yang digunakan untuk membentuk matriks A:

```
# membentuk matriks A
pred <- cbind(intercept=1, Girth=trees$Girth, Height=trees$Height)
head(A)
```

```
##       [,1] [,2] [,3]
## [1,]   27    6   -1
## [2,]    6   15    2
```

```
## [3,]    1    1   54
```

Langkah selanjutnya adalah membentuk matriks b. Berikut adalah sintaks yang digunakan:

```
resp<- trees$Volume
head(resp)
```

```
## [1] 10.3 10.3 10.2 16.4 18.8 19.7
```

Untuk memperoleh koefisien β, kita dapat mencarinya dengan cara menyelesaikan Persamaan (6.41). Berikut adalah sintaks yang digunakan:

```
A <- t(pred) %*% pred
b <- t(pred) %*% resp

Ab <- cbind(A,b)
(x <- gauss_jordan(Ab))
```

```
##           intercept Girth Height
## intercept         1     0      0 -57.9877
## Girth             0     1      0   4.7082
## Height            0     0      1   0.3393
```

Berdasarkan hasil yang diperoleh, persamaan linier yang terbentuk disajikan pada Persamaan (6.42).

$$Volume = 4.7081605Girth + 0.3392512Height - 57.9876589 \tag{6.42}$$

Pembaca juga dapat menggunakan fungsi lain untuk memperoleh nilai koefisien tersebut, seperti: `lu_solve()`dan`solve()`. untuk fungsi `jacobi()` dan `gauss_seidel()`, kita harus pastikan syarat-syarat terkait metode tersebut. Berikut adalah contoh penyelesaian menggunakan sintaks lainnya:

```
# metode LU
lu_solve(A,b)
```

```
## $P
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
##
```

```
## $L
##       [,1]  [,2] [,3]
## [1,]  1.00 0.000    0
## [2,] 13.25 1.000    0
## [3,] 76.00 1.054    1
##
## $U
##           intercept Girth Height
## intercept        31 410.7 2356.0
## Girth             0 295.4  311.5
## Height            0   0.0  889.6
##
## $result
##          [,1]
## [1,] -57.9877
## [2,]   4.7082
## [3,]   0.3393
```

```
# fungsi solve()
solve(A,b)
```

```
##              [,1]
## intercept -57.9877
## Girth       4.7082
## Height      0.3393
```

`R` juga menyediakan fungsi untuk membentuk model regresi linier. Fungsi yang digunakan adalah `lm()`. Berikut sintaks yang digunakan untuk membentuk model linier menggunakan fungsi `lm()`:

```
lm(Volume~Girth+Height, data=trees)
```

```
##
## Call:
## lm(formula = Volume ~ Girth + Height, data = trees)
##
## Coefficients:
## (Intercept)        Girth       Height
##     -57.988        4.708        0.339
```

6.6.2 Aliran Massa Dalam Reaktor

Pada *sub-chapter* ini penulis akan memberikan penerapan aljabar linier untuk menghitung konsentrasi suatu zat atau parameter lingkungan dalam reaktor yang saling terhubung. Pada contoh kasus kali ini diasumsikan terdapat lima buah reaktor yang saling terhubung satu sama lain sesuai Gambar 6.3. Debit air ($\frac{m^3}{detik}$) dan konsentrasi zat pencemar ($\frac{mg}{m^3}$) disajikan pula diagram alir tersebut. Diasumsikan kelima buah reaktor tersebut dalam kondisi *steady* dan volume reaktor diasumsikan sama. Kesetimbangan massa persatuan waktu dalam kondisi *steady* disajikan pada Persamaan (6.43).

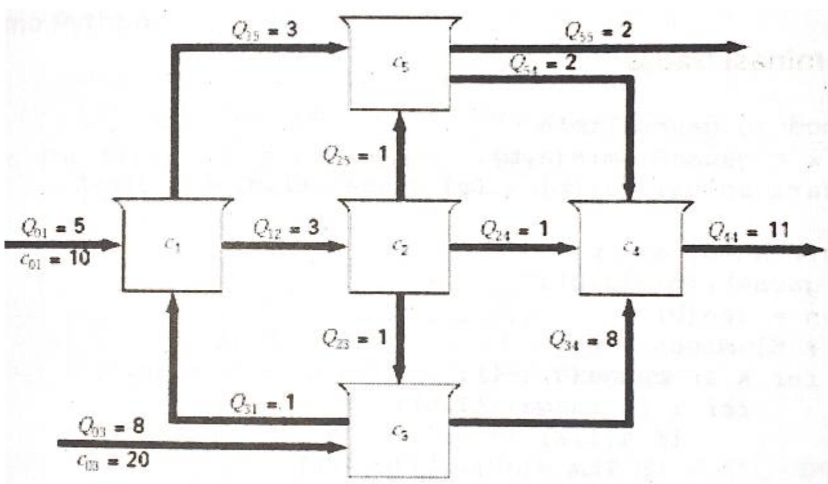

Figure 6.3: Aliran massa dalam reaktor.

$$m_i n = m_o ut \tag{6.43}$$

$$Q_{in} C_{in} = Q_{out} C_{out} \tag{6.44}$$

Berdasarkan Gambar 6.3, dapat dibentuk lima buah sistem persamaan linier. Persamaan linier yang terbentuk disajikan sebagai berikut:

$$\begin{gathered}
6c_1 - c_3 = 50 \\
-3c_1 + 3c_2 = 0 \\
-c_2 + 9c_3 = 160 \\
-c_2 - 8c_3 + 11c_4 - 2c_5 = 0 \\
-3c_1 - c_2 + 4c_5 = 0
\end{gathered}$$

Untuk menyelesaiakan sistem persamaan linier tersebut dan memperoleh nilai c dari masing-masing reaktor, kiat perlu mengubahnya dulu kedalam bentuk matriks $Ax = b$. Berikut adalah matriks yang terbentuk:

$$\begin{bmatrix} 6 & 0 & -1 & 0 & 0 \\ -3 & 3 & 0 & 0 & 0 \\ 0 & -1 & 9 & 0 & 0 \\ 0 & -1 & -8 & 11 & -2 \\ -3 & -1 & 0 & 0 & 4 \end{bmatrix} c = \begin{bmatrix} 50 \\ 0 \\ 160 \\ 0 \\ 0 \end{bmatrix}$$

Kita akan menyelesaikannya dengan menggunakan metode elminasi Gauss-Jordan, dekomposisi LU, iterasi Jacobi, dan iterasi Gauss-Seidel. Untuk dapat menyelesaikannya menggunakan metode-metode tersebut pada `R`, kita perlu membentuk matriksnya terlebih dahulu:

```
(A <- matrix(c(6,-3,0,0,-3,
               0,3,-1,-1,-1,
               -1,0,9,-8,0,
               0,0,0,11,0,
               0,0,0,-2,4),nrow=5))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    6    0   -1    0    0
## [2,]   -3    3    0    0    0
## [3,]    0   -1    9    0    0
## [4,]    0   -1   -8   11   -2
## [5,]   -3   -1    0    0    4
```

```
(b <- c(50,0,160,0,0))
```

```
## [1]  50   0 160   0   0
```

Metode Eliminasi Gauss-Jordan

```
gauss_jordan(cbind(A,b))
```

```
##                    b
## [1,] 1 0 0 0 0 11.51
## [2,] 0 1 0 0 0 11.51
## [3,] 0 0 1 0 0 19.06
## [4,] 0 0 0 1 0 17.00
## [5,] 0 0 0 0 1 11.51
```

Metode Dekomposisi LU

```
lu_solve(A,b)
```

```
## $P
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
##
## $L
##      [,1]    [,2]     [,3] [,4] [,5]
## [1,]  1.0  0.0000  0.00000    0    0
## [2,] -0.5  1.0000  0.00000    0    0
## [3,]  0.0 -0.3333  1.00000    0    0
## [4,]  0.0 -0.3333 -0.92453    1    0
## [5,] -0.5 -0.3333 -0.07547    0    1
##
## $U
##      [,1] [,2]       [,3] [,4] [,5]
## [1,]    6    0 -1.000e+00    0    0
## [2,]    0    3 -5.000e-01    0    0
## [3,]    0    0  8.833e+00    0    0
## [4,]    0    0  0.000e+00   11   -2
## [5,]    0    0  1.110e-16    0    4
##
## $result
## [1] 11.51 11.51 19.06 17.00 11.51
```

Metode Iterasi Jacobi

```
jacobi(A,b, maxiter=100)
```

```
## $X
##       [,1]
## [1,] 11.51
## [2,] 11.51
## [3,] 19.06
## [4,] 17.00
## [5,] 11.51
##
## $iter
## [1] 17
```

Metode Iterasi Gauss-Seidel

```
gauss_seidel(A,b, maxiter=200)
```

```
## $X
##        [,1]
## [1,] 11.51
## [2,] 11.51
## [3,] 19.06
## [4,] 17.00
## [5,] 11.51
##
## $iter
## [1] 7
```

Berdasarkan seluruh metode tersebut, diperoleh konsentrasi zat pencemar pada masing-masing reaktor adalah sebagai berikut:

$$
\begin{gathered}
c_1 = 11,50943\frac{mg}{m^3} \\
c_2 = 11,50943\frac{mg}{m^3} \\
c_3 = 19,05660\frac{mg}{m^3} \\
c_4 = 16,99828\frac{mg}{m^3} \\
c_5 = 11.50943\frac{mg}{m^3}
\end{gathered}
$$

6.7 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
2. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
3. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
4. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
5. Sanjaya, M. 2015. **Metode Numerik Berbasis Phython**. Penerbit Gava Media: Yogyakarta.
6. Suparno, S. 2008. **Komputasi untuk Sains dan Teknik Edisi II**. Departemen Fisika-FMIPA Universitas Indonesia.

6.8 Latihan

1. Selesaikan sistem persamaan linier berikut menggunakan eliminasi Gauss!

$$
\begin{aligned}
-4x + 4y &= -1 \\
-2x + 2y - 3z &= -3 \\
3x + 1y - 3z &= -3
\end{aligned}
$$

2. Carilah penyelesaian dari sistem persamaan linier soal no.1 menggunakan algoritma dekomposisi LU!
3. Tunjukan 5 iterasi pertama sistem persamaan linier berikut menggunakan algoritma Jacobi dan Gauss-Seidel!

$$
\begin{aligned}
3x + 2y - 1z &= -3 \\
-3x - 3y - 3z &= 9 \\
1y - 1z &= -1
\end{aligned}
$$

4. Gunakan fungsi `jacobi()` dan `gauss_seidel()` untuk menyelesaikan sistem persamaan linier pada soal no.4 dan tentukan metode mana yang paling cepat memperoleh penyelesaian? (**petunjuk**: gunakan fungsi `system.time()` dan jumlah iterasi yang diperlukan untuk memperoleh hasil yang konvergen)
5. Apakah yang terjadi jika kita menginputkan matriks segiempat A kedalam fungsi `solve()` dan apa yang akan terjadi jika selanjutnya argumen pada fungsi tersebut juga menyertakan vektor b?
6. Dengan menggunakan dataset `mtcars` buatlah persamaan linier variabel `mpg` sebagai fungsi dari variabel `wt`, `hp`, dan `qsec` menggunakan algoritma dekomposisi LU?

Chapter 7

Akar Persamaan Non-Linier

Persamaan non-linier dapat diartikan sebagai persamaan yang tidak mengandung syarat seperti persamaan linier, sehingga persamaan non-linier dapat merupakan:

a. Persamaan yang memiliki pangkat selain satu (misal: x^2)
b. Persamaan yang mempunyai produk dua variabel (misal: xy)

Dalam penyelesaian persamaan non-linier diperlukan akar-akar persamaan non-linier, dimana akar sebuah persamaan non-linier $f(x) = 0$ merupakan nilai x yang menyebabkan nilai $f(x)$ sama dengan nol. Dalam hal ini dapat disimpulkan bahwa akar-akar penyelesaian persamaan non-linier merupakan titik potong antara kurva $f(x)$ dengan sumbu x. Ilustrasi penjelasan tersebut ditampilkan pada Gambar 7.1.

Contoh sederhana dari penentuan akar persamaan non-linier adalah penentuan akar persamaan kuadratik. Secara analitik penentuan akar persamaan kuadratik dapat dilakukan menggunakan Persamaan (7.1).

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4a}}{2a} \tag{7.1}$$

Untuk masalah yang lebih rumit, penyelesaian analitik sudah tidak mungkin dilakukan. Metode numerik dapat digunakan untuk menyelesaikan masalah yang lebih kompleks. Untuk mengetahui apakah suatu persamaan non-linier memiliki akar-akar penyelesaian atau tidak, diperlukan analisa menggunakan Teorema berikut:

Teorema 7.1 (root)**.** *Suatu range x=[a,b] mempunyai akar bila f(a) dan f(b) berlawanan tanda atau memenuhi f(a).f(b)<0*

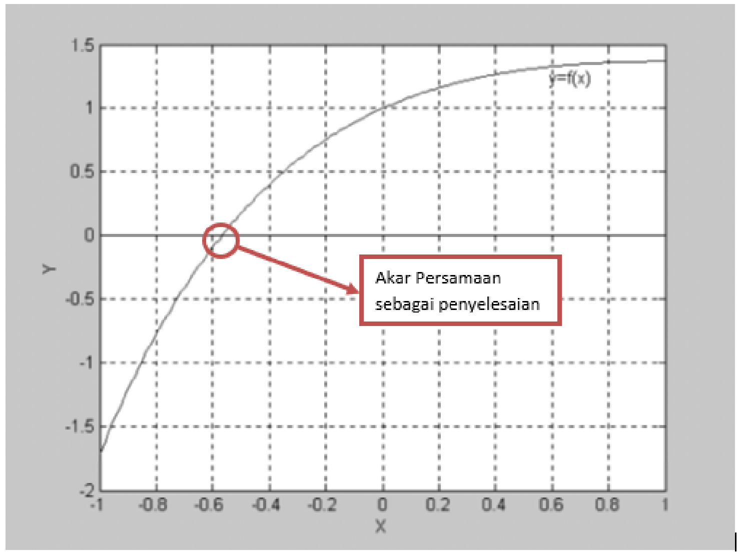

Figure 7.1: Penyelesaian persamaan non-linier.

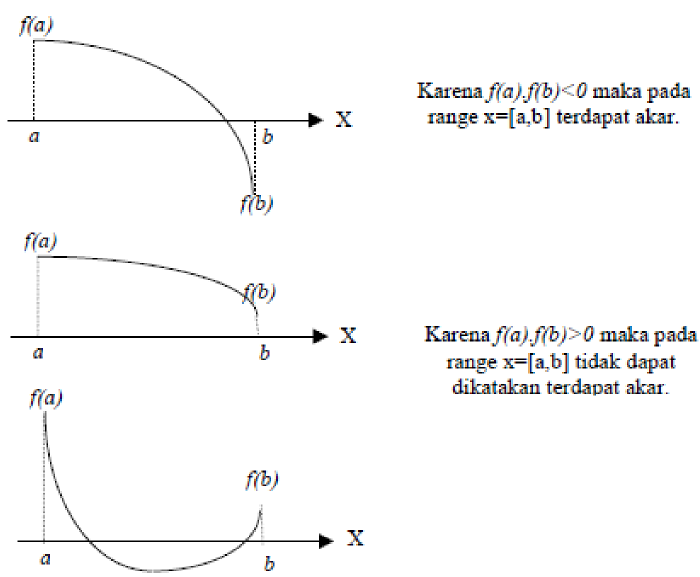

Figure 7.2: Ilustrasi teorema Bolzano.

Untuk memahami teorema tersebut perhatikan ilustrasi pada Gambar 7.2.

Pada Chapter 7 ini, akan dilakukan sejumlah pembahasan antara lain:

- penentuan akar persamaan dengan metode tertutup
- penentuan akar persamaan dengan metode terbuka
- fungsi-fungsi `R` untuk mementukan akar persamaan non-linier
- studi kasus

7.1 Metode Tertutup

Metode tertutup disebut juga metode *bracketing*. Disebut sebagai metode tertutup karena dalam pencarian akar-akar persamaan non-linier dilakukan dalam suatu selang $[a, b]$.

7.1.1 Metode Tabel

Penyelesaian persamaan non-linier menggunakan metode tabel dilakukan dengan membagi persamaan menjadi beberapa area, dimana untuk $x = [a, b]$ dibagi sebanyak N bagian dan pada masing-masing bagian dihitung nilai $f(x)$ sehingga diperoleh nilai $f(x)$ pada setian N bagian.

Bila nilai $f(x_k) = 0$ atau mendekati nol, dimana $a \leq k \leq b$, maka dikatakan bahwa x_k adalah penyelesaian persamaan $f(x)$. Bila tidak ditemukan, dicari nilai $f(x_k)$ dan $f(x_{k+1})$ yang berlawanan tanda. Bila tidak ditemukan, maka persamaan tersebut dapat dikatakan tidak mempunyai akar untuk rentang $[a, b]$.

Bila akar persamaan tidak ditemukan, maka ada dua kemungkinan untuk menentukan akar persamaan, yaitu:

a. Akar persamaan ditentukan oleh nilai mana yang lebih dekat. Bila $f(x_k) \leq f(x_{k+1})$, maka akarnya x_k. Bila $f(x_{k+1}) \leq f(x_k)$, maka akarnya x_{k+1}.
b. Perlu dicari lagi menggunakan rentang $x = [x_k, x_{k+1}]$.

Secara grafis penyelesaian persamaan non-linier menggunakan metode table disajikan pada Gambar 7.3.

Algoritma Metode Tabel

1. Definisikan fungsi $f(x)$

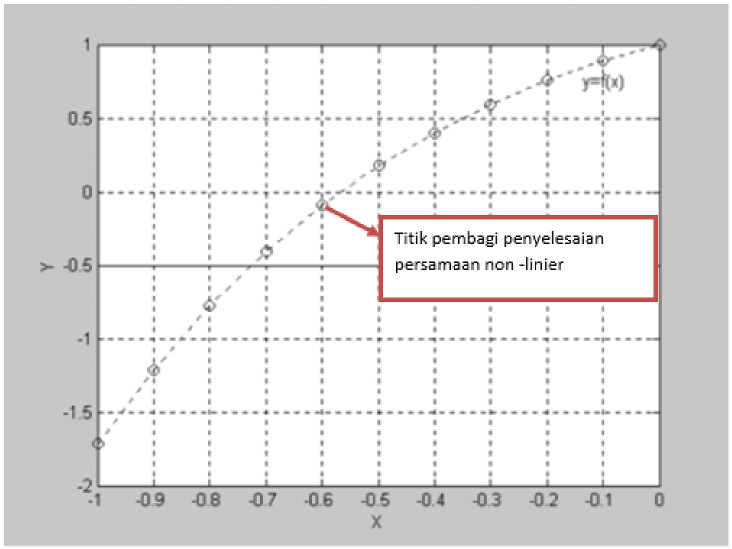

Figure 7.3: Ilustrasi metode tabel.

2. Tentukan rentang untuk x yang berupa batas bawah a dan batas atas b.
3. Tentukan jumlah pembagi N
4. Hitung step pembagi

$$h = \frac{b + a}{N} \tag{7.2}$$

5. Untuk $i = 0$ s/d N, hitung:

$$x_i = a + i.h \tag{7.3}$$

$$y_i = f(x_i) \tag{7.4}$$

6. Untuk $i = 0$ s/d N, dimana

- Bila $f(x) = 0$, maka akarnya x_k
- Bila $f(a) f(b) < 0$, maka:
  - $f(x_k) \leq f(x_{k+1})$, maka akarnya x_k
  - Bila tida, x_{k+1} adalah penyelesaian atau dapat dikatakan penyelesaian berada diantara x_k dan x_{k+1}.

Table 7.1: Penyelesaian persamaan x+exp(x)=0

x	fx
-1.0	-0.6321
-0.9	-0.4934
-0.8	-0.3507
-0.7	-0.2034
-0.6	-0.0512
-0.5	0.1065
-0.4	0.2703
-0.3	0.4408
-0.2	0.6187
-0.1	0.8048
0.0	1.0000

Kita dapat membuat suatu fungsi pada `R` untuk melakukan proses iterasi pada metode Tabel. Fungsi `root_table()` akan melakukan iterasi berdasarkan step algoritma 1 sampai 5. Berikut adalah sintaks yang digunakan:

```
root_table <- function(f, a, b, N=20){
    h <- abs((a+b)/N)
    x <- seq(from=a, to=b, by=h)
    fx <- rep(0, N+1)
    for(i in 1:(N+1)){
      fx[i] <- f(x[i])
    }
    data <- data.frame(x=x, fx=fx)
    return(data)
}
```

Contoh 7.1. Carilah akar persamaan $f(x) = x + e^x$ pada rentang $x = [-1, 0]$?

Jawab:

Sebagai permulaan, jumlah pembagi yang digunakan adalah $N = 10$. Dengan menggunakan fungsi `root_table()` diperoleh hasil yang disajikan pada Tabel 7.1.

```
tabel <- root_table(f=function(x){x+exp(x)},
                     a=-1, b=0, N=10)
```

Berdasarkan Tabel 7.1 diperoleh penyelesaian di antara $-0,6$ dan $-0,5$ dengan nilai $f(x)$ masing-masing sebesar $-0,0512$ dan $-0,1065$, sehingga dapat diambil penyelesaian $x = -0,6$. Kita dapat terus melakukan iterasi sampai memperoleh nilai $f(x) <$ nilai toleransi dengan terus merubah rentang yang diberikan. Iterasi berikutnya dengan nilai pembagi sama dan rentang nilai $x = [-0,6; -0,5]$ diperoleh nilai $x = -0,57$ dan $f(x) = 0,00447$.

Untuk melihat gambaran lokasi akar, kita dapat pulang mengeplotkan data menggunakan fungsi plot. Berikut adalah fungsi yang digunakan:

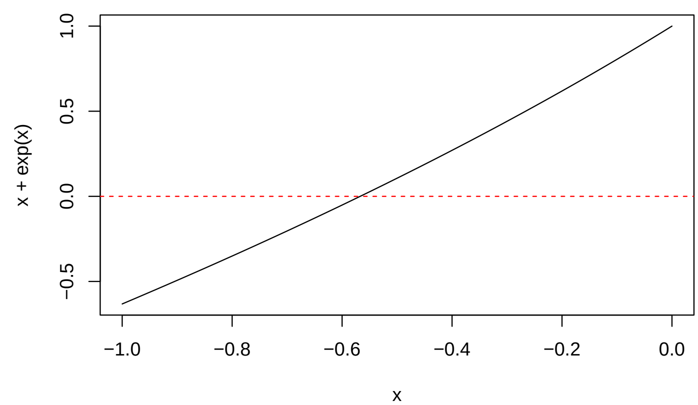

Figure 7.4: Plot fungsi x+exp(x) pada rentang -1 sampai 0.

Untuk mengetahui lokasi akar dengan lebih jelas, kita dapat memperkecil lagi rentang nilai yang dimasukkan dalam fungsi `curve()`.

Metode tabel pada dasarnya memiliki kelemahan yaitu cukup sulit untuk mendapatkan error penyelesaian yang cukup kecil, sehingga metode ini jarang sekali digunakan untuk menyelesaikan persamaan non-linier. Namun, metode ini cukup baik digunakan dalam menentukan area penyelesaian sehingga dapat dijadikan acuan metode lain yang lebih baik.

7.1.2 Metode Biseksi

Prinsip metode bagi dua adalah mengurung akar fungsi pada interval $x = [a, b]$ atau pada nilai x batas bawah a dan batas atas b. Selanjutnya interval tersebut terus menerus dibagi 2 hingga sekecil mungkin, sehingga nilai hampiran yang dicari dapat ditentukan dengan tingkat toleransi tertentu. Untuk lebih memahami metode biseksi, perhatikan visualisasi pada Gambar 7.5.

Metode biseksi merupakan metode yang paling mudah dan paling sederhana dibanding metode lainnya. Adapun sifat metode ini antara lain:

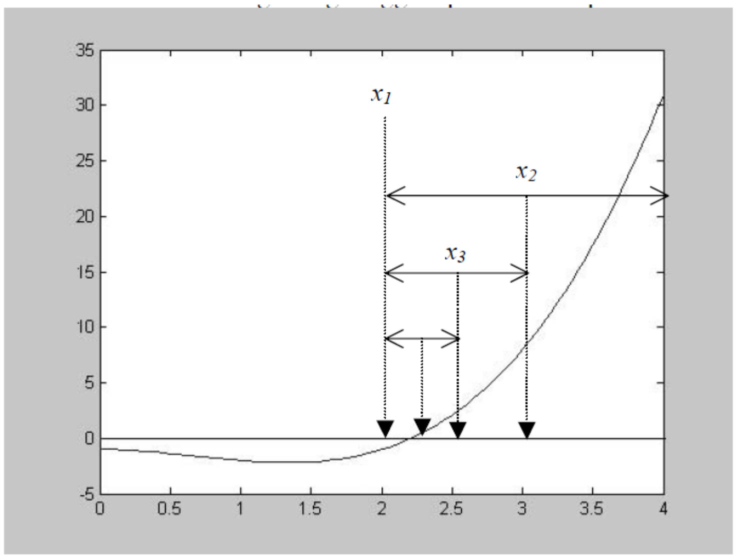

Figure 7.5: Ilustrasi metode biseksi.

1. Konvergensi lambat
2. Caranya mudah
3. Tidak dapat digunakan untuk mencari akar imaginer
4. Hanya dapat mencari satu akar pada satu siklus.

Algoritma Metode Biseksi

1. Definisikan fungsi $f(x)$
2. Tentukan rentang untuk x yang berupa batas bawah a dan batas atas b.
3. Tentukan nilai toleransi e dan iterasi maksimum N
4. Hitung $f(a)$ dan $f(b)$
5. Hitung:

$$x = \frac{a+b}{2} \tag{7.5}$$

6. Hitung $f(x)$
7. Bila $f(x).f(a) < 0$, maka $b = x$ dan $f(b) = f(x)$. Bila tidak, $a = x$ dan $f(a) = f(x)$
8. Bila $|b - a| < e$ atau iterasi maksimum maka proses dihentikan dan didapatkan akar=x, dan bila tidak ulangi langkah 6.

9. Jika sudah diperoleh nilai dibawah nilai toleransi, nilai akar selanjutnya dihitung berdasarkan Persamaan (7.5) dengan nilai a dan b merupakan nilai baru yang diperoleh dari proses iterasi.

Berdasarkan algoritma tersebut, kita dapat menyusun suatu fungsi pada `R` yang dapat digunakan untuk melakukan iterasi tersebut. Fungsi `root_bisection()` merupakan fungsi yang telah penulis susun untuk melakukan iterasi menggunakan metode biseksi. Berikut adalah sintaks dari fungsi tersebut:

```
root_bisection <- function(f, a, b, tol=1e-7, N=100){
  iter <- 0
  fa <- f(a)
  fb <- f(b)

  while(abs(b-a)>tol){
    iter <- iter+1
    if(iter>N){
      warning("iterations maximum exceeded")
      break
    }
    x <- (a+b)/2
    fx <- f(x)
    if(fa*fx>0){
      a <- x
      fa <- fx
    } else{
      b <- x
      fb <- fx
    }
  }

  # iterasi nilai x sebagai return value
  root <- (a+b)/2
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.2. Carilah akar persamaan $f(x) = xe^{-x}+1$ pada rentang $x = [-1, 0]$ dengan nilai toleransi sebesar 10^-7?

Jawab:

Langkah pertama dalam penghitungan adalah menghitung nilai x menggunakan Persamaan (7.5).

$$x = \frac{-1+0}{2} = -0,5$$

Hitung nilai $f(x)$ dan $f(a)$.

$$f(x) = -0,5.e^{0,5} + 1 = 0,175639$$

$$f(a) = -1.e^{1} + 1 = -1,71828$$

Berdasarkan hasil perhitungan diperoleh:

$$f(x).f(a) < 0$$

Sehingga $b = x$ dan $f(b) = f(x)$. Iterasi dilakukan kembali dengan menggunakan nilai b tersebut.

Untuk mempersingkat waktu iterasi kita akan menggunakan fungsi `root_bisection()` pada `R`. Berikut adalah sintaks yang digunakan:

```
root_bisection(function(x){x*exp(-x)+1},
               a=-1, b=0)
```

```
## $`function`
## function(x){x*exp(-x)+1}
## <bytecode: 0x000001dbe908b4a0>
##
## $root
## [1] -0.5671
##
## $iter
## [1] 24
```

Berdasarkan hasil iterasi diperoleh akar persamaan $x = -2.980232e - 08$ dan iterasi yang diperlukan untuk memperolehnya sebanyak 24 iterasi.

7.1.3 Metode Regula Falsi

Metode regula falsi merupakan metode yang menyerupai metode biseksi, dimana iterasi dilakukan dengan terus melakukan pembaharuan rentang untuk memperoleh akar persamaan. Hal yang membedakan metode ini dengan metode biseksi adalah pencarian akar didasarkan pada slope (kemiringan) dan selisih tinggi dari kedua titik rentang. Titik pendekatan pada metode regula-falsi disajikan pada Persamaan (7.6).

$$x = \frac{f(b).a - f(a).b}{f(b) - f(a)} \tag{7.6}$$

Ilustrasi dari metode regula falsi disajikan pada Gambar 7.6.

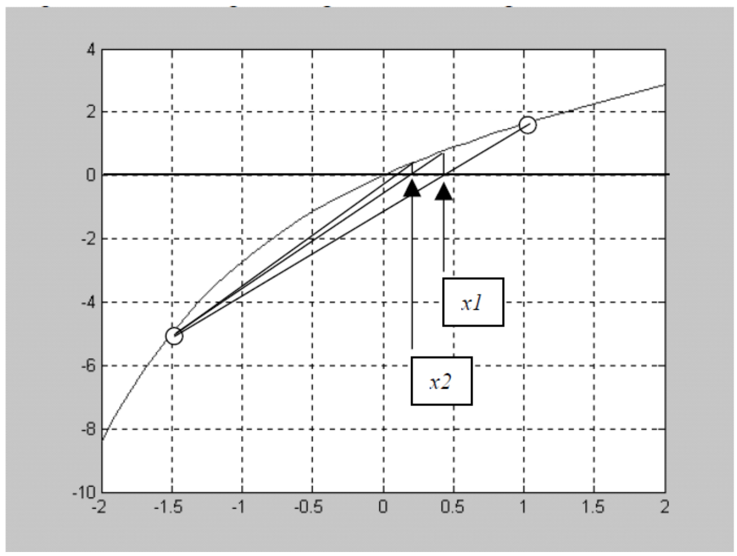

Figure 7.6: Ilustrasi metode regula falsi.

Algoritma Metode Regula Falsi

1. Definisikan fungsi $f(x)$
2. Tentukan rentang untuk x yang berupa batas bawah a dan batas atas b.
3. Tentukan nilai toleransi e dan iterasi maksimum N
4. Hitung $f(a)$ dan $f(b)$
5. Untuk iterasi $i = 1$ s/d N

- Hitung nilai x berdasarkan Persamaan (7.6)
- Hitung $f(x)$
- Hitung $error = |f(x)|$
- Jika $f(x).f(a) < 0$, maka $b = x$ dan $f(b) = f(x)$. Jika tidak, $a = x$ dan $f(a) = f(x)$.

6. Akar persamaan adalah x

Fungsi `root_rf()` didasarkan pada langkah-langkah di atas. Sintaks fungsi tersebut adalah sebagai berikut:

```
root_rf <- function(f, a, b, tol=1e-7, N=100){
  iter <- 1
  fa <- f(a)
  fb <- f(b)
  x <- ((fb*a)-(fa*b))/(fb-fa)
  fx <- f(x)

  while(abs(fx)>tol){
    iter <- iter+1
    if(iter>N){
      warning("iterations maximum exceeded")
      break
    }
    if(fa*fx>0){
      a <- x
      fa <- fx
    } else{
      b <- x
      fb <- fx
    }
    x <- (fb*a-fa*b)/(fb-fa)
    fx <- f(x)
  }

  # iterasi nilai x sebagai return value
  root <- x
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.3. Selesaikan persamaan non-linier pada Contoh 7.2 menggunakan metode regula falsi pada rentang $x = [-1, 0]$ dengan nilai toleransi sebesar 10^-7?

Jawab:

Langkah pertama penyelesaian dilakukan dengan mencari nilai $f(a)$ dan $f(b)$.

$$f(a) = -1.e^1 + 1 = -1,71828$$
$$f(b) = 0.e^0 + 1 = 1$$

Hitung nilai x dan $f(x)$.

$$x = \frac{(1. - 1) - (-1,71828.0)}{1 + 1,71828} = -0.36788$$

$$f(x) = -0.36788.e^{0.36788} + 1 = 0.468536$$

Berdasarkan hasil perhitungan diperoleh:

$$f(x).f(a) < 0$$

Sehingga $b = x$ dan $f(b) = f(x)$. Iterasi dilakukan kembali dengan menggunakan nilai b tersebut.

Untuk mempercepat proses iterasi, kita dapat pula menggunakan fungsi `root_rf()` pada `R`. Berikut adalah sintaks yang digunakan:

```
root_rf(function(x){x*exp(-x)+1},
               a=-1, b=0)
```

```
## $`function`
## function(x){x*exp(-x)+1}
## <bytecode: 0x000001dbea09b988>
##
## $root
## [1] -0.5671
##
## $iter
## [1] 15
```

Berdasarkan hasil perhitungan diperoleh nilai $x = -0,5671433$ dan jumlah iterasi yang diperlukan adalah 15. Jumlah ini lebih sedikit dari jumlah iterasi yang diperlukan pada metode iterasi biseksi yang juga menunjukkan metode ini lebih cepat memperoleh persamaan dibandingkan metode biseksi.

7.2 Metode Terbuka

Metode terbuka merupakan metode yang menggunakan satu atau dua tebakan awal yang tidak memerlukan rentang sejumlah nilai. Metode terbuka terdiri dari beberapa jenis yaitu metode iterasi titik tetap, metode Newton-Raphson, dan metode Secant.

7.2.1 Metode Iterasi Titik Tetap

Metode iterasi titik tetap merupakan metode penyelesaian persamaan non-linier dengan cara menyelesaikan setiap variabel x yang ada dalam suatu persamaan dengan sebagian yang lain sehingga diperoleh $x = g(x)$ untuk masing-masing variabel x. Sebagai contoh, untuk menyelesaikan persamaan $x + e^x = 0$, maka persamaan tersebut perlu diubah menjadi $x = e^x$ atau $g(x) = e^x$. Secara grafis metode ini diilustrasikan seperti Gambar 7.7.

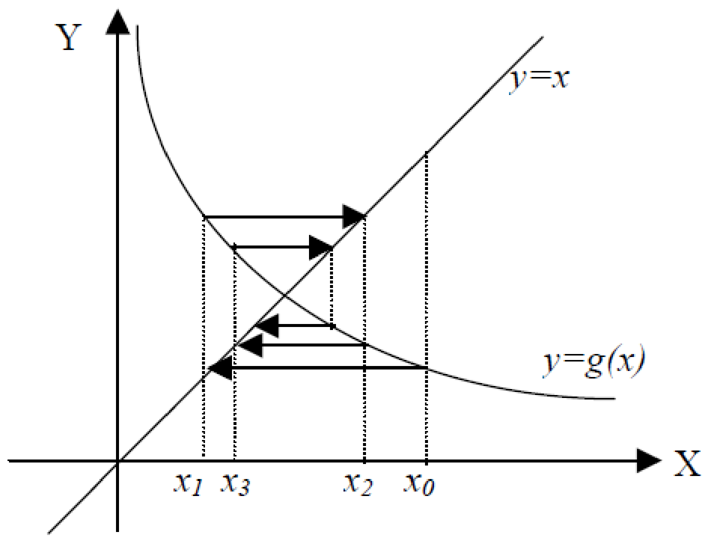

Figure 7.7: Ilustrasi metode iterasi titik tetap.

Algoritma Metode Iterasi Titik Tetap

1. Definisikan $f(x)$ dan $g(x)$
2. Tentukan nilai toleransi e dan iterasi masimum (N)
3. Tentukan tebakan awal x_0
4. Untuk iterasi $i = 1$ s/d N atau $f(x_i terasi) \geq e \rightarrow x_i = g(x_{i-1})$, Hitung $f(x_i)$
5. Akar persamaan adalah x terakhir yang diperoleh

FUngsi `root_fpi()` dapat digunakan untuk melakukan iterasi dengan argumen fungsi berupa persamaan non-linier, nilai tebakan awal, nilai toleransi, dan jumlah iterasi maksimum. Berikut adalah sintaks fungsi tersebut:

```
root_fpi <- function(f, x0, tol=1e-7, N=100){
  iter <- 1
  xold <- x0
  xnew <- f(xold)

  while(abs(xnew-xold)>tol){
    iter <- iter+1
    if(iter>N){
      stop("No solutions found")
    }
    xold <- xnew
    xnew <- f(xold)
  }

  root <- xnew
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.4. Selesaikan persamaan non-linier pada Contoh 7.2 menggunakan metode iterasi titik tetap?

Jawab:

Untuk menyelesaikan persamaan non-linier tersebut kita perlu mentransformasi persamaan non-linier tersebut terlebih dahulu.

$$xe^{-x} + 1 = 0 \ \rightarrow x = -\frac{1}{e^{-x}}$$

Untuk tebakan awal digunakan nilai $x = -1$

$$x_1 = -\frac{1}{e^1} = -2,718282$$

Nilai x tersebut selanjutnya dijadikan nilai input pada iterasi selanjutnya:

$$x_2 = -\frac{1}{e^{2,718282}} = -0,06598802$$

iterasi terus dilakukan sampai diperoleh $|x_{i+1} - x_i| \leq e$.

Untuk mempercepat proses iterasi kita dapat menggunakan bantuan fungsi `root_fpi()`. Berikut adalah sintaks yang digunakan:

```
root_fpi(function(x){-1/exp(-x)}, x0=-1)
```

```
## $`function`
## function(x){-1/exp(-x)}
## <bytecode: 0x000001dbe5412500>
##
## $root
## [1] -0.5671
##
## $iter
## [1] 29
```

Berdasarkan hasil iterasi diperoleh nilai $x = -0,5671433$ dengan jumlah iterasi yang diperlukan sebanyak 29 kali. Jumlah iterasi akan bergantung dengan nilai tebakan awal yang kita berikan. Semakin dekat nilai tersebut dengan akar, semakin cepat nilai akar diperoleh.

7.2.2 Metode Newton-Raphson

Metode Newton-Raphson merupakan metode penyelesaian persamaan non-linier dengan menggunakan pendekatan satu titik awal dan mendekatinya dengan memperhatikan slope atau gradien. titik pendekatan dinyatakan pada Persamaan (7.7).

$$x_{n+1} = x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right)} \tag{7.7}$$

Ilustrasi metode Newton-Raphson disajikan pada Gambar 7.8.

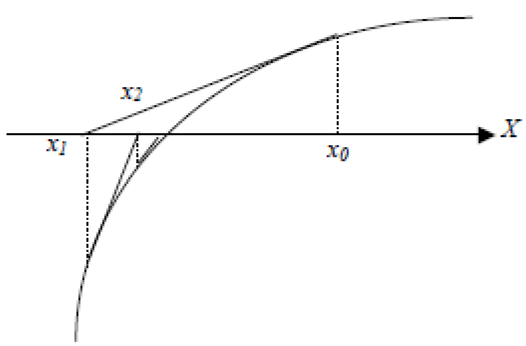

Figure 7.8: Ilustrasi metode Newton-Raphson.

Algoritma Metode Newton-Raphson

1. Definisikan $f(x)$ dan $f'(x)$
2. Tentukan nilai toleransi e dan iterasi masimum (N)
3. Tentukan tebakan awal x_0
4. Hitung $f(x_0)$ dan $f'(x_0)$
5. Untuk iterasi $i = 1$ s/d N atau $|f(x)| \geq e$, hitung x menggunakan Persamaan (7.7)
6. Akar persamaan merupakan nilai x_i terakhir yang diperoleh.

Fungsi `root_newton()` merupakan fungsi yang dibuat menggunakan algoritma di atas. Fungsi tersebut dituliskan pada sintaks berikut:

```
root_newton <- function(f, fp, x0, tol=1e-7, N=100){
  iter <- 0
  xold<-x0
  xnew <- xold + 10*tol

  while(abs(xnew-xold)>tol){
    iter <- iter+1
    if(iter>N){
      stop("No solutions found")
    }
    xold<-xnew
    xnew <- xold - f(xold)/fp(xold)
  }

  root<-xnew
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.5. Selesaikan persamaan non-linier $x - e^{-x} = 0$ menggunakan metode Newton-Raphson?

Jawab:

Untuk dapat menggunakan metode Newton-Raphson, terlebih dahulu kita perlu memperoleh turunan pertama dari persamaan tersebut.

$$f(x) = x - e^{-x} \rightarrow f'(x) = 1 + e^{-x}$$

Tebakan awal yang digunakan adalah $x = 0$.

$$f(x_0) = 0 - e^{-0} = -1$$

$$f'(x_0) = 1 + e^{-0} = 2$$

Hitung nilai x baru:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0 - \frac{-1}{2} = 0,5$$

Untuk mempercepat proses iterasi, kita dapat menggunakan fungsi `root_newton()`. Berikut adalah sintaks yang digunakan:

```
root_newton(function(x){x-exp(-x)},
            function(x){1+exp(-x)},
              x0=0)
```

```
## $`function`
## function(x){x-exp(-x)}
## <bytecode: 0x000001dbe6fca528>
##
## $root
## [1] 0.5671
##
## $iter
## [1] 5
```

Berdasarkan hasil iterasi diperoleh akar penyelesaian persamaan non-linier adalah $x = 0,5671433$ dengan jumlah iterasi yang diperlukan adalah 5 iterasi.

Dalam penerapannya metode Newton-Raphson dapat mengalami kendala. Kendala yang dihadapi adalah sebagai berikut:

1. titik pendekatan tidak dapat digunakan jika merupakan titik ekstrim atau titik puncak. Hal ini disebabkan pada titik ini nilai $f'(x) = 0$. Untuk memahaminya perhatikan ilustasi yang disajikan pada Gambar 7.9. Untuk menatasi kendala ini biasanya titik pendekatan akan digeser.

2. Sulit memperoleh penyelesaian ketika titik pendekatan berada diantara 2 titik stasioner. Untuk memahami kendala ini perhatikan Gambar 7.10. Untuk menghindarinya, penentuan titik pendekatan dapat menggunakan bantuan metode tabel.

3. Turunan persamaan sering kali sulit untuk diperoleh (tidak dapat dikerjakan dengan metode analitik).

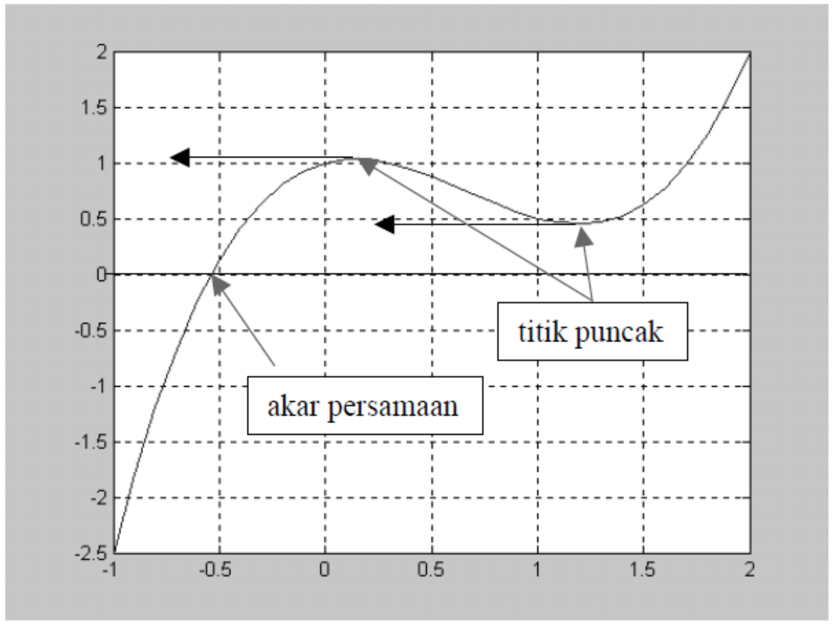

Figure 7.9: Ilustrasi titik pendekatan di titik puncak.

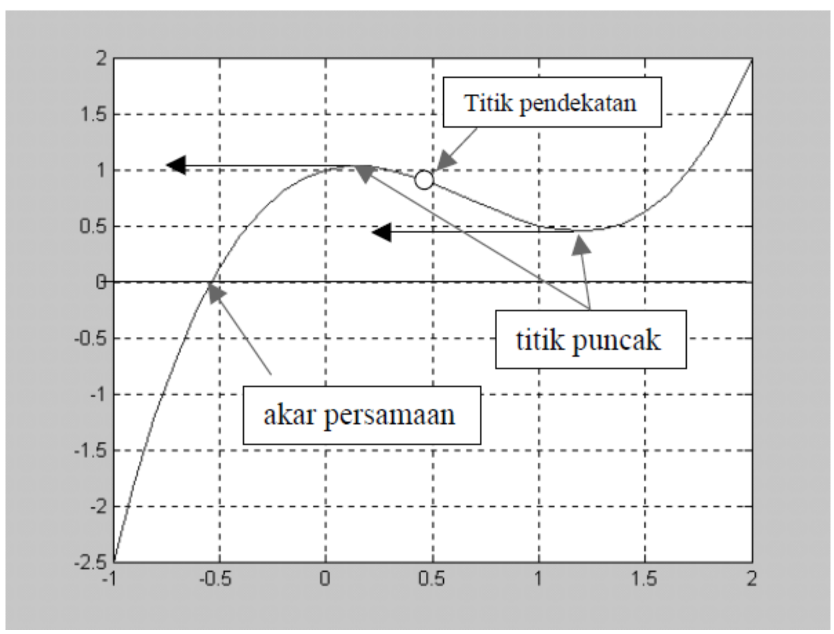

Figure 7.10: Ilustrasi titik pendekatan diantara 2 titik stasioner.

7.2.3 Metode Secant

Metode Secant merupakan perbaikan dari metode regula-falsi dan Newton Raphson, dimana kemiringan dua titik dinyatakan secara diskrit dengan mengambil bentuk garis lurus yang melalui satu titik. Persamaan yang dihasilkan disajikan pada Persamaan (7.8).

$$y - y_0 = m\left(x - x_0\right) \tag{7.8}$$

Nilai m merupakan transformasi persamaan tersebut.

$$m_n = \frac{f\left(x_n\right) - f\left(x_{n-1}\right)}{x_n - x_{n-1}} \tag{7.9}$$

Bila $y = f\left(x\right)$ dan y_n dan x_n diketahui, maka titik ke $n+1$ adalah:

$$y_{n+1} - y_n = m_n\left(x_{n+1} - x_n\right) \tag{7.10}$$

Bila titik x_{n+1} dianggap akar persamaan maka nilai $y_{n+1} = 0$, sehingga diperoleh:

$$-y_n = m_n\left(x_{n+1} - x_n\right) \tag{7.11}$$

$$\frac{m_n x_n - y_n}{m_n} = x_{n+1} \tag{7.12}$$

atau

$$x_{n+1} = x_n - y_n \frac{1}{m_n} \tag{7.13}$$

$$x_{n+1} = x_n - f\left(x_n\right) \frac{x_n - x_{n+1}}{f\left(x_n\right) - f\left(x_{n+1}\right)} \tag{7.14}$$

Berdasarkan Persamaan (7.14) diketahui bahwa untuk memperoleh akar persamaan diperlukan 2 buah titik pendekatan. Dalam buku ini akan digunakan titik pendekatan kedua merupakan titik pendekatan pertama ditambah sepuluh kali nilai toleransi.

$$x_1 = x_0 + 10 * tol \tag{7.15}$$

Algoritma Metode Secant

1. Definisikan $f(x)$ dan $f'(x)$
2. Tentukan nilai toleransi e dan iterasi masimum (N)
3. Tentukan tebakan awal x_0 dan x_1
4. Hitung $f(x_0)$ dan $f(x_1)$
5. Untuk iterasi $i = 1$ s/d N atau $|f(x)| \geq e$, hitung x menggunakan Persamaan (7.14)
6. Akar persamaan adalah nilai x yang terakhir.

Fungsi `root_secant()` merupakan fungsi yang penulis buat untuk melakukan iterasi menggunakan metode Secant. Berikut merupakan sintaks dari fungsi tersebut:

```
root_secant <- function(f, x, tol=1e-7, N=100){
  iter <- 0

  xold <- x
  fxold <- f(x)
  x <- xold+10*tol

  while(abs(x-xold)>tol){
    iter <- iter+1
    if(iter>N)
      stop("No solutions found")

    fx <- f(x)
    xnew <- x - fx*((x-xold)/(fx-fxold))
    xold <- x
    fxold <- fx
    x <- xnew
  }

  root<-xnew
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.6. Selesaikan persamaan non-linier pada Contoh 7.5 menggunakan metode Secant?

Jawab:

Untuk menyelesaikan persamaan tersebut digunakan nilai pendekatan awal $x_0 = 0$ dan $x_1 = 0 + 10 * 10^{-7} = 10^{-6}$.

$$f\left(x_0\right) = 0 - e^{-0} = -1$$

$$f\left(x_1\right) = 10^{-6} - e^{-10^{-6}} = -0,999998$$

Hitung nilai x_2 dan $f\left(x_2\right)$.

$$x_2 = 0 + 0,999998 \frac{10^{-6} - 0}{-0,999998 + 1} = 0,499999$$

Untuk mempercepat proses iterasi kita dapat menggunakan fungsi `root_secant()` pada `R`. Berikut sintaks yang digunakan:

```
root_secant(function(x){x-exp(-x)}, x=0)
```

```
## $`function`
## function(x){x-exp(-x)}
## <bytecode: 0x000001dbe9868f70>
##
## $root
## [1] 0.5671
##
## $iter
## [1] 6
```

Berdasarkan hasil iterasi diperoleh nilai akar penyelesaian adalah $x = 0,5671433$ dengan iterasi dilakukan sebanyak 6 kali.

Secara umum metode Secant menawarkan sejumlah keuntungan dibanding metode lainnya. Pertama, seperti metode Newton-Raphson dan tidak seperti metode tertutup lainnya, metode ini tidak memerlukan rentang pencarian akar penyelesaian. Kedua, tidak seperti metode Newton-Raphson, metode ini tidak memerlukan pencarian turunan pertama persamaan non-linier secara analitik, dimana tidak dapat dilakukan otomasi pada setiap kasus.

Adapun kerugian dari metode ini adalah berpotensi menghasilkan hasil yang tidak konvergen sama seperti metode terbuka lainnya. Selain itu, kecepatan konvergensinya lebih lambat dibanding metode Newton-Raphson.

7.3 Penyelesaian Persamaan Non-Linier Menggunakan Fungsi `uniroot` dan `uniroot.all`

Paket `base` pada `R` menyediakan fungsi `uniroot()` untuk mencari akar persamaan suatu fungsi pada rentang spesifik. Fungsi ini menggunakan metode

Brent yaitu kombinasi antara *root bracketing*, biseksi, dan interpolasi invers kuadrat. Format fungsi tersebut secara sederhana adalah sebagai berikut:

```
uniroot(f, interval, tol=.Machine$double.eps^0.25,
        maxiter=1000)
```

Catatan:

- **f**: persamaan non-linier
- **interval**: vektor interval batas bawah dan atas
- **tol**: nilai toleransi
- **maxiter**: iterasi maksimum

Berikut adalah contoh penerapan fungsi `uniroot()`:

```
uniroot(function(x){x*exp(-x)+1},
        interval=c(-1,0), tol=1e-7)
```

```
## $root
## [1] -0.5671
##
## $f.root
## [1] 1.533e-08
##
## $iter
## [1] 7
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 5e-08
```

Berdasarkan hasil iterasi diperoleh akar persamaan tersebut adalah $-0,5671433$ dengan jumlah iterasi sebanyak 7 iterasi dan tingkat presisi sebesar $5e-08$.

Fungsi lain yang dapat digunakan untuk mencari akar persamaan adalah `uniroot.all()` dari paket `rootSolve`. Fungsi ini mengatasi kelemahan dari `uniroot()`, dimana `uniroot()` tidak bekerja jika fungsi hanya menyentuh dan tidak melewati sumbu nol $y = 0$. Untuk memahaminya perhatikan contoh berikut:

```
uniroot(function(x){sin(x)+1}, c(-pi,0))
```

Bandingkan dengan sintaks berikut:

```
uniroot(function(x){sin(x)+1}, c(-pi,-pi/2))
```

```
## $root
## [1] -1.571
##
## $f.root
## [1] 0
##
## $iter
## [1] 0
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 0
```

Untuk menggunakan fungsi `uniroot.all()`, jalankan sintaks berikut:

```
library(rootSolve)
```

Jalankan kembali fungsi dan rentang di mana `uniroot()` tidak dapat bekerja:

```
uniroot.all(function(x){sin(x)+1}, c(-pi,0))
```

```
## [1] -1.571
```

7.4 Akar Persamaan Polinomial Menggunakan Fungsi `polyroot`

Fungsi `polyroot()` pada paket `base` dapat digunakan untuk memperoleh akar dari suatu polinomial. Algortima yang digunakan dalam fungsi tersebut adalah algoritma Jenkins dan Traub.

Untuk dapat menggunakannya kita hanya perlu memasukkan vektor koefisien dari polinomial. Pengisian elemen dalam vektor dimulai dari variabel dengan pangkat tertinggi menuju variabel dengan pangkat terendah. Berikut adalah contoh bagaimana fungsi `polyroot()` digunakan untuk mencari akar polinomial $f\left(x\right)=x^2+1$:

```
polyroot(c(1,0,1))
```

```
## [1] 0+1i 0-1i
```

Contoh lainnya adalah mencari akar polinomial $f(x) = 4x^2 + 5x + 6$:

```
polyroot(c(4,5,6))
```

```
## [1] -0.4167+0.7022i -0.4167-0.7022i
```

Pembaca dapat mencoba membuktikan hasil yang diperoleh tersebut menggunakan metode analitik.

7.5 Studi Kasus

Penerapan penyelesaian sistem persamaan non-linier banyak dijumpai dalam berbagai kasus di bidang lingkungan. Pada bagian ini penulis tidak akan menjelaskan seluruhnya. Penulis hanya akan menjelaskan penerapannya pada sebuah persamaan yaitu Hukum Bernoulli.

7.5.1 Persamaan Van Der Walls

7.5.2 Hukum Bernoulli

Misalkan terdapat sebuah saluran dengan penampang sesuai dengan Gambar 7.11.

Berdasarkan hukum Bernoulli, maka diperoleh persamaan berikut:

$$\frac{Q^2}{2gb^2h_0^2} + h_0 = \frac{Q^2}{2gb^2h^2} + h + H \tag{7.16}$$

Persamaan tersebut dapat dilakukan transformasi menjadi persamaan berikut:

$$f(h) = h^3 + \left(H - \frac{Q^2}{2gb^2h_0^2} - h_0\right)h^2 + \frac{Q^2}{2gb^2} = 0 \tag{7.17}$$

Data-data terkait saluran tersebut adalah sebagai berikut:

- $Q = 1,2\ \frac{m^3}{\text{det}}$ = volume aliran fluida tiap satuan waktu
- $g = 9,81\ \frac{m}{s^2}$ =percepatan gravitasi

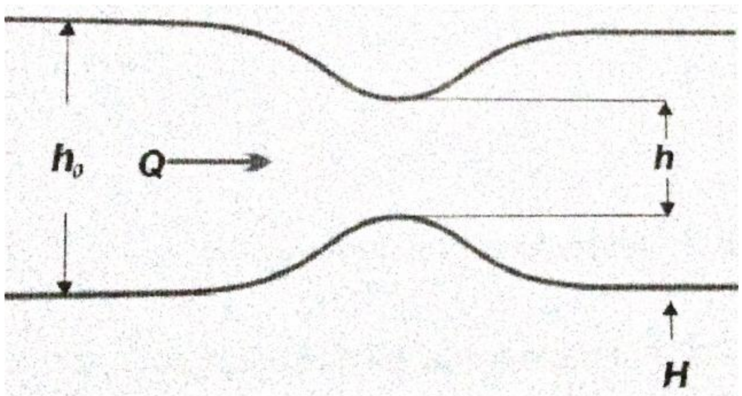

Figure 7.11: Aliran fluida pada sebuah pipa.

- $b = 1,8\ m$ =lebar pipa
- $h_0 = 0,6\ m$ =ketinggian air maksimum
- $H = 0,075\ m$ =tinggi pelebaran pipa
- h = ketinggian air

Kita dapat menggunakan pendekatan numerik untuk menentukan h. Pada studi kasus ini tidak dijelaskan lokasi dimana akar penyelesaian berada, sehingga metode terbuka seperti Secant cukup sesuai untuk menyelesaikannya:

Berikut adalah persamaan yang baru setelah seluruh data dimasukkan kedalam tiap variabelnya:

$$f(h) = h^3 + \left(0,075 - \frac{1,2^2}{2 \times 9,81 \times 1,8^2 \times 0,6^2} - 0,6\right) h^2 + \frac{1,2^2}{2 \times 9,81 \times 1,8^2} = 0$$

Untuk penyelesaiannya penulis akan memberikan tebakan awal nilai $h = h_0 = 0,6$. Berikut adalah sintaks penyelesaian menggunakan metode secant:

```
f <- function(h){
  (h^3) + ((0.075-((1.2^2)/(2*9.81*(1.8^2)*(0.6^2))))*h^2)+ (1.2^2/(2*9.81*(1.8^2)))
}
root_secant(f, 0.6)
```

```
## $`function`
## function(h){
##   (h^3) + ((0.075-((1.2^2)/(2*9.81*(1.8^2)*(0.6^2))))*h^2)+ (1.2^2/(2*9.81*(1.8^2)))
## }
## <bytecode: 0x000001dbe68b12d0>
```

```
##
## $root
## [1] -0.287
##
## $iter
## [1] 26
```

Berdasarkan hasil perhitungan diperoleh nilai $h = -0,2870309$ atau ketinggian air sekitar $0,3\ m$ dengan jumlah iterasi sebanyak 26 kali.

Pembaca dapat mencoba menggunakan metode lain seperti metode tertutup. Untuk dapat melakukannya, pembaca perlu memperoleh rentang lokasi akar persamaan tersebut berada menggunakan metode tabel.

7.6 Referensi

1. Atmika, I.K.A. 2016. **Diktat Mata Kuliah: Metode Numerik**. Jurusan Teknik Mesin Universitas Udayana.
2. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
3. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
4. Jones, O. Maillardet, R. Robinson, A. 2014. **Introduction to Scientific Programming and Simulation Using R**. CRC Press
5. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
6. Sanjaya, M. 2015. **Metode Numerik Berbasis Phython**. Penerbit Gava Media: Yogyakarta.
7. Sudiadi dan Teguh R. 2015. **Metode Numerik**. STMIK

7.7 Latihan

1. Temukan akar persamaan dari persamaan non-linier $f(x) = x^3 - 2x + 2$ menggunakan metode terbuka dengan $x_0 = 0$ dan $x_0 = \frac{1}{2}$!
2. Apakah kelebihan dari metode tertutup (contoh: metode biseksi) dibanding metode terbuka (contoh: Newton-Raphson)? (**catatan**: pembaca dapat pula mencari dari referensi lainnya)
3. Temukan akar persamaan dari persamaan $f(x) = \frac{sin(x)}{x}$ dengan rentang pencarian $x = 0,5$ dan $x = 1$!
4. Pada kondisi apakah metode Secant lebih dipilih dibanding metode Newton-Raphson?

5. Modifikasilah fungsi `root_bisection()` dan `root_rf()` sehingga kita tidak perlu memasukkan argumen `a` dan `b` dan hanya perlu memasukkan satu vektor `interval` kedalam fungsi tersebut! (**contoh**: `interval=c(a,b)`)

Chapter 8

Interpolasi dan Ekstrapolasi

Pada dunia nyata, data sering kali tidak tersaji secara lengkap. Seringkali terdapat nilai data yang hilang (*missing value*). Terdapat banyak penyebab dari kondisi tersebut, baik akibat kesalahan manusianya maupun keterbatasan kemampuan alat ukur.

Kondisi lain yang muncul dari data yang kita miliki adalah adanya *outlier* atau nilai yang berbeda jauh dengan mayoritas data yang kita miliki. Nilai tersebut akan menentukan hasil analisis atau uji statistik yang kita lakukan, terlebih lagi jika uji statistik yang kita lakukan menggunakan metode parametrik.

Terdapat banyak cara untuk menangani kondisi-kondisi tersebut. Sejumlah peneliti memilih untuk menghapus data tersebut. Hal ini dapat dilakukan jika jumlah data yang kita miliki cukup besar. Bagaimana jika data yang kita miliki sedikit dan pengukuran ulang cukup mahal atau cukup sulit dilakukan?. Salah satu cara yang dapat dilakukan adalah dengan melakukan interpolasi terhadap data.

Interpolasi dan ekstrapolasi adalah proses "menebak" nilai data dengan memperhatikan data lain yang kita miliki. Interpolasi merupakan teknik untuk mencari nilai suatu variabel yang hilang pada rentang data yang diketahui, sedangkan ektrapolasi merupakan teknik menemukan nilai suatu variabel diluar rentang data yang telah diketahui. Data lain yang kita miliki seringkali memiliki sejumlah pola. Pola yang terbentuk dapat berupa polinomial atau mengelompok. Tiap pola akan memiliki metode pendekatan yang berbeda-beda. Terdapat kemungkinan tak terbatas dari pola data tersebut. Penilaian profesional atau ahli diperlukan untuk menentukan metode mana yang sesuai berdasarkan riwayat penelitian atau pekerjaan yang pernah dilakukan sebelumnya.

Pada Chapter 8 penulis akan menjelaskan teknik-teknik interpolasi yang dapat kita lakukan. Adapun yang akan dibahas pada *Chapter* ini adalah sebagai berikut:

- Teknik interpolasi polinomial
- Teknik interpolasi piecewise
- Studi kasus penerapan teknik interpolasi

8.1 Interpolasi Polinomial

Interpolasi polinomial merupakan teknik interpolasi dengan mengasumsikan pola data yang kita miliki mengikuti pola polinomial baik berderajat satu (linier) maupun berderajat tinggi. Interpolasi dengan metode ini dilakukan dengan terlebih dahulu membentuk persamaan polinomial. Persamaan polinomial yang terbentuk selanjutnya digunakan untuk melakukan interpolasi dari nilai yang diketahui atau ekstrapolasi (prediksi) dari nilai diluar rentang data yang diketahui.

Pada Chapter 8.1 pembahasan akan dibagi menjadi 3 bagian. Bagian pertama kita akan mengulang kembali teknik evaluasi polinomial, sedangkan dua bagian selanjutnya akan membahas teknik interpolasi linier dan polinomial orde tinggi dengan menjadikan pembahasan bagian pertama sebagai dasar pada dua bagian berikutnya.

8.1.1 Mengevaluasi Polinomial

Pada *Chapter* ini pembaca akan mempelajari teknik untuk melakukan substitusi nilai x pada persamaan polinomial untuk memperoleh nilai y. Terdapat berbagai pendekatan dalam melakukan proses tersebut, mulai dari metode naive maupun metode Horner. Kedua metode akan menghasilkan hasil yang sama namun dengan proses komputasi yang berbeda. Metode naive cenderung lambat dalam proses komputasi karena jumlah proses yang dilakukan dalam sekali proses lebih banyak dari pada metode Horner.

Untuk memahami metode-metode evaluasi polinomial yang telah disebutkan tersebut, secara umum persamaan polinomial disajikan pada Persamaan (8.1).

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \tag{8.1}$$

dimana a merupakan koefisien polinomial, x merupakan variabel, dan n merupakan indeks dan pangkat polinomial.

Pada metode naive kita melakukan evaluasi polinomial sama dengan cara kita melakukan evaluasi polinomial saat kita SMA. Nilai x akan disubstitusikan pada masing-masing elemen persamaan polinomial. Masing-masing elemen polinomial selanjutnya dijumkahkan untuk menghitung y.

Pada `R` kita dapat menuliskan sebuah fungsi untuk melakukan evaluasi polinomial menggunakan metode naive tersebut. Pada fungsi tersebut, koefisien polinomial akan disimpan kedalam sebuah vektor dengan urutan pengisian mulai dari koefisien dengan pangkat x terendah ke tertinggi.

```
naive_poly <- function(x, coeff){
  n <- length(x)
  y <- rep(0, n)

  for(i in 1:length(coeff)){
    y <- y + coeff[i]*(x^(i-1))
  }

  return(y)
}
```

Contoh 8.1. Hitung nilai y pada persamaan $f(x) = x^4 + 3x^3 - 15x^2 - 19x + 30$, jika diketahui nilai x adalah -1, 0, dan 1!

Jawab:

Untuk dapat menghitung nilai y menggunakan fungsi `naive_poly()` pada persamaan tersebut dengan nilai x yang diketahui, kita perlu merubah koefisien persamaan tersebut dan nilai x yang diketahui menjadi vektor:

```
x <- c(-1,0,1)
coeff <- c(30,-19,-15,3,1)
```

Masukkan vektor-vektor yang telah terbentuk tersebut kedalam fungsi `naive_poly()`.

```
naive_poly(x, coeff)
```

```
## [1] 32 30  0
```

Berdasarkan hasil perhitungan diperoleh nilai y masing-masing sebesar 32, 30, dan 0. Pembaca dapat mengeceknya sendiri hasil perhitungan tersebut menggunakan cara manual.

Kita dapat meningkatkan efisiensi proses perhitungan pada fungsi `naive_poly()` tersebut. Sebagai contoh, setiap kali kita melakukan loop untuk menghitung nilai y pada polinomial, kita dapat memperoleh eksponensial dari x. Namun untuk setiap koefisien a_i, nilai eksponensial yang terkait x_i merupakan seri produk.

$$x^i = \prod_{j=0}^{i} x \tag{8.2}$$

Untuk x^i, terdapat sebanyak i koefisien x dikalikan bersamaan. Namun, untuk setiap x^{i-1} terdapat lebih sedikit 1 perkalian dibanding koefisien x dengan pangkat yang lebih besar dan seterusnya.

Berdasarkan ilustrasi tersebut, kita dapat membentuk fungsi `better_poly()` sebagai perbaikan dari fungsi `naive_poly()`. Berikut adalah sintaks yang digunakan:

```
better_poly <- function(x, coeff){
  n <- length(x)
  y <- rep(0, n)
  cached_x <- 1

  for(i in 1:length(coeff)){
    y <- y + coeff[i]*cached_x
    cached_x <- cached_x * x
  }
  return(y)
}
```

Contoh 8.2. Hitung nilai y pada persamaan yang disajikan pada Contoh 8.1 menggunakan nilai x yang telah diketahui pada soal tersebut menggunakan fungsi `better_poly()`?

Jawab:

```
better_poly(x, coeff)
```

```
## [1] 32 30  0
```

Sejauh ini kita telah membentuk 2 fungsi yaitu `naive_poly()` dan `better_poly()`. Kedua fungsi tersebut memiliki perbedaan proses menghitung yang mempengaruhi efisiensinya masing-masing. Sebagai contoh jika diberikan polinomial berderajat 10, fungsi `naive_poly()` akan mengejakan perkalian dalam proses *loop* sebanyak 55 kali, sedangkan fungsi `better_poly()` akan melakukannya sebanyak 20 kali ($2n$ perkalian).

Metode lain yang lebih efisien dalam melakukan evaluasi polinomial adalah metode Horner. Metode ini oleh William Horner pada abad ke-18. Dalam metode Horner, bentuk polinomial pada Persamaan (8.1) akan ditransformasi menjadi Persamaan (8.3).

$$\begin{aligned} f(x) &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \\ &= a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n \\ &= a_0 + x\left(a_1 + \cdots + a_{n-1} x^{n-2} + a_n x^{n-1}\right) \\ &= a_0 + x\left(a_1 + \cdots + x\left(a_{n-1} + x\left(a_n\right)\right)\cdots\right) \end{aligned} \tag{8.3}$$

Berdasarkan Persamaan (8.3), jika kita melakukan perhitungan pada persamaan polinomial berderajat 10, kita dapat mereduksi perhitungan menjadi 10 perkalian dan 10 penjumlahan. Jumlah tersebut sangat kecil dibandingkan kedua metode sebelumnya dan dapat dikatakan lebih efisien dibandingkan metode lainnya.

FUngsi `horner_poly()` merupakan fungsi yang dibentuk bedasarkan Persamaan (8.3). Sintaks fungsi tersebut adalah sebagai berikut:

```
horner_poly <- function(x, coeff){
  n <- length(x)
  y <- rep(0, n)

  for(i in length(coeff):1){
    y <- coeff[i] + x * y
  }
  return(y)
}
```

Contoh 8.3. Kerjakan kembali Contoh 8.2 fungsi `horner_poly()`?

Jawab:

```
horner_poly(x, coeff)
```

```
## [1] 32 30  0
```

8.1.2 Interpolasi Linier

Misalkan kita memiliki 3 buah data dengan dua buah variabel kita misalkan variabel x dan variabel y. Pada salah satu data terdapat data yang hilang pada. Agar ketiga data tersebut tetap dapat digunakan dalam iterasi diperlukan interpolasi untuk "menebak" nilai dari data yang hilang.

Berdasarkan pengukuran yang sebelumnya pernah dilakukan diketahui bahwa pola data yang terbentuk variabel x dan y divisualisasikan menggunakan scatterplot adalah pola linier. Berdasarkan hal tersebut interpolasi dilakukan dengan menggunakan metode linier.

Interpolasi linier dilakukan dengan terlebih dahulu membentuk fungsi linier. Dengan kata lain kita perlu mencari nilai slope m dan *intercept* b. Nilai m dihitung sebagai rasio selisih jarak dua titik pada sumbu y dan sumbu x yang dapat dituliskan melalui Persamaan (8.4).

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{8.4}$$

Nilai *intercept* (titik potong pada sumbu y) dihitung menggunakan Persamaan (8.5).

$$b = y_2 - mx_2 \tag{8.5}$$

Algoritma Interpolasi Linier

1. Tentukan dua buah titik (x, y) sebagai dasar pembentukan persaman linier.
2. Hitung m menggunakan Persamaan (8.4)
3. Hitung b menggunakan Persamaan (8.5)
4. Definiskan fungsi linier berdasarkan nilai m dan b
5. Hitung y dengan cara substitusi nilai x pada persamaan linier untuk melakukan interpolasi atau ekstrapolasi nilai y yang ingin dicari.

Algoritma poin 1 sampai 4 tersebut, kita dapat membentuk fungsi pembentuk persamaan linier dari 2 titik yang diketahui. Fungsi tersebut disajikan pada sintaks berikut:

```
linear_inter <- function(x, y){
  m <- (y[2]-y[1]) / (x[2]-x[1])
  b <- y[2] - m*x[2]

  return(c(b, m))
}
```

Contoh 8.4. Diketahui koordinat 2 buah titik yaitu (0,-1) dan (2,3). Jika diketahui titik ketiga memiliki koordinat sumbu x sebesar 1. Lakukan interpolasi untuk menentukan koordinat sumbu y titik ketiga tersebut!

Jawab:

Berdasarkan data-data yang terdapat pada soal terserbut, kita dapat menghitung nilai m dan b. Nilai m dapat dihitung sebagai berikut:

$$m = \frac{-1-3}{0-2} = 2$$

Dengan menggunakan nilai m tersebut kita dapat menghitung nilai b.

$$b = 3 - 2 * 3 = -1$$

Berdasarkan hasil perhitungan diperoleh persamaan linier yang terbentuk adalah sebagai berikut:

$$y = 2x - 1$$

Berdasarkan persamaan linier tersebut nilai y dapat dihitung.

$$y = 2 * 1 - 1 = 1$$

Kita dapat pula membentuk persamaan linier menggunakan fungsi `linear_inter()`. Berikut adalah sintaks yang digunakan:

```
x <- c(0,2)
y <- c(-1,3)

(coeff <- linear_inter(x,y))
```

```
## [1] -1  2
```

Setelah diperoleh koefisien persamaan linier berdasarkan dua titik tersebut, kita akan menggunakan fungsi `horner_poly()` untuk memperoleh nilai y. Berikut adalah sintaks yang digunakan:

```
horner_poly(1, coeff)
```

```
## [1] 1
```

Hasil interpolasi yang diperoleh dapat dikatakan sesuai dengan lokasi kedua titik data yang ditunjukkan pada Gambar 8.1. Berdasarkan hal tersebut, kita dapat yakin bahwa hasil interpolasi yang telah kita lakukan telah sesuai.

Metode interpolasi linier dapat dibilang merupakan metode interpolasi yang sangat sederhana. Disamping kemudahannya, metode ini memiliki potensi error numerik jika jarak antara kedua titik cukup berdekatan terlebih lagi jika selisih penyebut $(x_2 - x_1)$ sangat kecil sehingga akan menghasilkan nilai y yang sangat besar.

Disamping adanya potensi error numerik tersebut, metode ini menjadi dasar bagi metode interpolasi lain yang lebih kompleks. Metode selanjutnya merupakan pengembangan dari metode interpolasi ini.

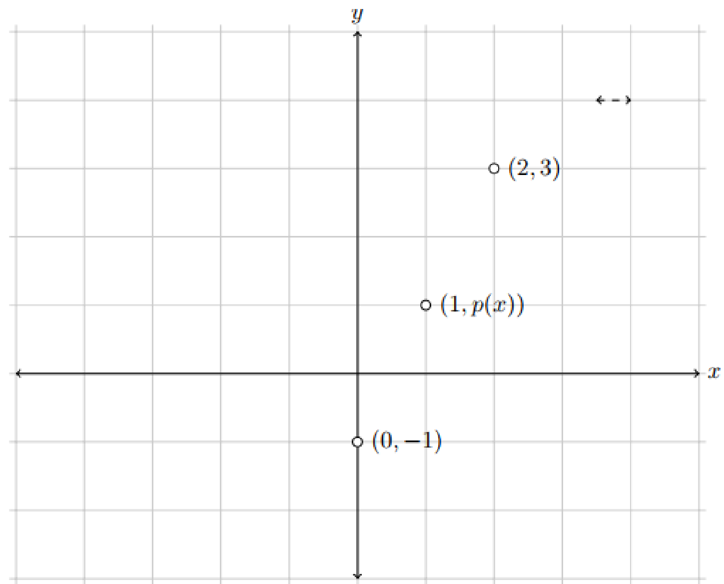

Figure 8.1: Interpolasi linier dua titik (Sumber:Howard, 2017).

8.1.3 Interpolasi Polinomial Orde Tinggi

Dengan menggunakan dua titik, kita dapat membentuk garis lurus (linier) yang tepat pada dua titik tersebut. Masalah timbul jika selisih nilai x kedua titik tersebut sangat kecil atau kedua titik tersebut memiliki nilai x yang sama. Hal ini akan menyebabkan slope yang dihasilkan menjadi tidak terhingga atau garis yang terbentuk adalah garis vertikal tegak lurus.

Bagaimana jika terdapat tiga buah titik? apakah kita masih bisa menggunakan interpolasi linie?. Ya, asalkan ketiga titik tersebut membentuk pola linier atau terletak pada satu garis yang sama. Pada kenyataannya kondisi tersebut jarang terjadi, sehingga pendekatan menggunakan polinomial orde lebih tinggi diperlukan. Persamaan kuadratik (polinomial orde dua) dapat digunakan untuk membentuk persamaan polinomial pada ketiga titik tersebut, sehingga iterasi dapat dilakukan. Untuk 4 buah titik data, polinomial orde tiga dapat digunakan untuk melakukan interpolasi. Secara umum berdasarkan penjelasan tersebut, untuk n titik data interpolasi dapat dilakukan menggunakan persamaan polinomial orde $n-1$.

Diberikan set data berpasangan yang telah diurutkan (x_i, y_i), fungsi interpolasi harus memenuhi persyaratan berikut:

$$p(x_i) = y_i \tag{8.6}$$

Untuk setiap i. Sebagai tambahan, fungsi interpolasi berupa fungsi polinomial dengan bentuk umum sebagai berikut:

$$y_i = \beta_n x_i^n + \beta_{n-1} x_i^{n-1} + \cdots + \beta_1 x_i + \beta_0 \tag{8.7}$$

Persamaan (8.7) dapat dituliskan kedalam bentuk matriks yang ditampilkan pada Persamaan (8.8).

$$\begin{bmatrix} x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n^n & x_n^{n-1} & \cdots & x_n & 1 \end{bmatrix} \begin{bmatrix} \beta_n \\ \beta_{n-1} \\ \vdots \\ \beta_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{8.8}$$

Persamaan matrik tersebut dapat dituliskan sebagai $X\beta = y$. Untuk menyelesaikan persamaan tersebut (memperoleh nilai β), pembaca dapat membaca kembali Chapter 6. Matriks X disebut sebagai matriks Vandemonde dan matriks tersebut mengandung sejumlah nilai x dengan pangkat sampai dengan n.

Algoritma Interpolasi Polinomial Orde Tinggi

1. Tentukan set titik berpasangan (x, y) yang telah diurutkan.
2. Bentuk matriks Vandermonde sesuai dengan Persamaan (8.8).
3. Definiskan persamaan matriks $X\beta = y$
4. Selesaikan persamaan matriks pada poin 3 untuk memperoleh nilai β
5. Definisikan persamaan polinomial berdasarkan koefisien β yang diperoleh
6. Lakukan substitusi x persamaan polinomial pada poin 5 untuk memperoleh nilai y

---

Berdasarkan algoritma poin 1 sampai 5, kita dapat membentuk suatu fungsi untuk membentuk persamaan polinomial berdasarkan Persamaan (8.8). Berikut adalah sintaks fungsi tersebut:

```
poly_inter <- function(x, y){
  if(length(x) != length(y))
    stop("Lenght of x and y vectors must be the same")

  n <- length(x)-1
  vandermonde <- rep(1, length(x))
```

```
  for(i in 1:n){
    xi <- x^i
    vandermonde <- cbind(vandermonde, xi)
  }
  beta <- solve(vandermonde, y)

  names(beta) <- NULL
  return(beta)
}
```

Contoh 8.5. Diketahui koordinat 3 buah titik yaitu (-1,-2), (1,2) dan (0,1). Jika diketahui titik keempat memiliki koordinat sumbu x sebesar -2. Lakukan interpolasi untuk menentukan koordinat sumbu y titik keempat tersebut!

Jawab:

Untuk menyelesaiakn contoh soal tersebut, kita perlu terlebih dahulu membentuk matriks sesuai dengan Persamaan (8.8). Berdasarkan soal tersebut, terdapat tiga buah titik data yang diketahui, sehingga polinomial yang hendak dibentuk selanjutnya adalah polinomial berderajat 2.

$$\begin{bmatrix} -1^2 & -1^1 & 1 \\ 1^2 & 1^1 & 1 \\ 0^2 & 0^1 & 1 \end{bmatrix} \begin{bmatrix} \beta_2 \\ \beta_1 \\ \beta_0 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}$$

Setelah matriks tersebut terbentuk, pembaca dapat menyelesaikannya menggunakan berbagai metode yang telah penulis jelaskan pada Chapter 6 untuk memperoleh nilai β.

Untuk menyelesaikan contoh soal tersebut pada `R`, kiat perlu membentuk matriks x dan y terlebih dahulu.

```
x <- c(-1, 1, 0)
y <- c(-2, 2, -1)
```

Koefisien persamaan polinomial dihitung menggunakan fungsi `poly_inter()`.

```
(coeff <- poly_inter(x, y))
```

```
## [1] -1  2  1
```

Berdasarkan hasil perhitungan, diperoleh nilai β. Nilai tersebut selanjutnya digunakan untuk membentuk persamaan polinomial. Berikut merupakan persamaan polinomial yang terbentuk:

$$f(x) = 1x^2 + 2x - 1$$

Fungsi `horner_poly()` selanjutnya digunakan untuk mengevaluasi polinomial tersebut. Berikut adalah hasil substitusi x pada persamaan tersebut:

```
horner_poly(-2, coeff)
```

```
## [1] -1
```

Hasil yang diperoleh terlihat cukup sesuai jika kita perhatikan visualisasi ketiga titik tersebut pada Gambar 8.2.

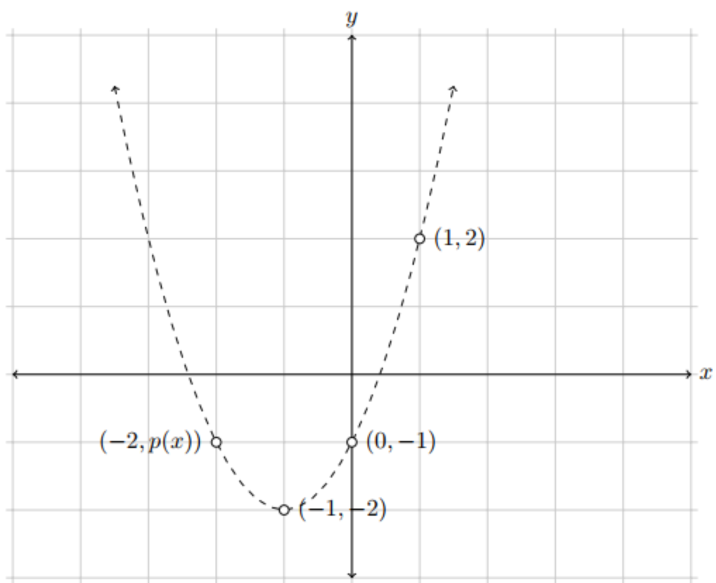

Figure 8.2: Interpolasi kuadratik tiga titik (Sumber:Howard, 2017).

Contoh 8.6. Dengan menggunakan data pada Contoh 8.4, lakukan proses perhitungan untuk membentuk persamaan polinomial menggunakan fungsi `poly_inter()`

Jawab:

Berdasarkan data pada Contoh 8.4, polinomial yang terbentuk merupakan polinomial derajat 1 (linier). Berikut adalah nilai koefisien yang dihasilkan dari perhitungan menggunakan fungsi `poly_inter()`.

```
x <- c(2, 0)
y <- c(3, -1)
poly_inter(x, y)
```

```
## [1] -1  2
```

Meskipun proses perhitungan menggunakan fungsi `poly_inter()` lebih rumit dibandingkan dengan fungsi `linear_poly()`, hasil perhitungan keduanya menggunakan data pada Contoh 8.4 menghasilkan hasil yang sama.

8.2 Interpolasi Piecewise

Interpolasi dengan polinomial sering memberikan hasil yang tidak dapat diterima. Interpolasi polinomial yang dihasilkan dari sejumlah besar data titik biasanya berderajat tinggi. Polinomial berderajat tinggi pada umumnya bersifat osilatif (grafiknya naik turun secara cepat). Akibatnya, perubahan data pada interval kecil dapat menyebabkan fluktuasi besar pada keseluruhan interval. Karena alasan ini, biasanya interpolasi hanya menggunakan polinomial berderajat rendah.

Interpolasi piecewise menawarkan alternatif lain. Pada interpolasi piecewise, pada titik yang berbeda sepanjang kurva, nilai fungsi lebih mungkin lebih baik didekati menggunakan dua atau lebih interpolasi. Pada metode ini kita akan membuat fungsi interpolasi ditiap antara dua titik observasi.

Pada sub-Chapter ini akan dijelaskan 2 buah metode interpolasi piecewise, yaitu: interpolasi linier piecewise dan interpolasi kubik spline. Interpolasi pertama dilakukan menggunakan persamaan linier, sehingga kurva yang terbentuk bukan merupakan kurva kontinu. Interpolasi selanjutnya dilakukan menggunakan persamaan polinomial berderajat tinggi sehingga kurva yang dihasilkan lebih halus (tidak ada sudut siku pada setiap titik).

8.2.1 Interpolasi Linier Piecewise

Interpolasi linier piecewise merupakan interpolasi yang menggunakan pendekatan interpolasi linier. Fungsi linier akan dibentuk pada setiap dua titik observasi. Untuk lebih memahaminya perhatikan kembali Gambar 8.2. Pada gambar tersebut sebelumnya kita telah membentuk persamaan kudratik untuk menghubungkan titik-titik tersebut. Dibanding menggunakan persamaan polinomial seperti kuadratik tersebut, interpolasi piecewise akan menghubungkan tiap dua titik observasi tersebut dengan garis lurus.

Algoritma Interpolasi Linier Piecewise

1. Tentukan set titik berpasangan (x, y) yang telah diurutkan berdasarkan nilai sumbu x.
2. Hitung m pada setiap dua titik berdekatan menggunakan Persamaan (8.4)
3. Hitung b pada setiap dua titik berdekatan menggunakan Persamaan (8.5)
4. Definiskan fungsi linier berdasarkan nilai m dan b
5. Hitung y dengan cara substitusi nilai x pada persamaan linier untuk melakukan interpolasi nilai y yang ingin dicari.
6. Untuk melakukan ekstrapolasi dengan titik observasi diluar rentang titik diketahui, gunakan persamaan linier yang berada pada bagian ujung terdekat dengan nilai x yang hendak dicari nilai y-nya.

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi pada `R` untuk membentuk persamaan linier piecewise. Berikut adalah sintaks yang digunakan:

```
pwise_linterp <- function(x, y){
  n <- length(x)-1

  y <- y[order(x)]
  x <- x[order(x)]

  m_vec <- b_vec <- c()

  for(i in 1:n){
    m <- (y[i+1]-y[i]) / (x[i+1]-x[i])
    b <- y[i+1] - m*x[i+1]
    m_vec <- c(m_vec, m)
    b_vec <- c(b_vec, b)
  }

  return(list(b = b_vec, m = m_vec))
}
```

Contoh 8.7. Tentukan persamaan-persamaan linier yang dihasilkan dari titik observasi yang ditampilkan pada Contoh 8.5?

Jawab:

Untuk menentukan persamaan-persamaan linier yang menghubungkan setiap titik, kita akan menggunakan fungsi `pwise_linterp()` yang telah kita buat. Berikut adalah sintaks yang digunakan:

```
x <- c(-1, 1, 0, -2)
y <- c(-2, 2, -1, -1)

pwise_linterp(x, y)
```

```
## $b
## [1] -3 -1 -1
##
## $m
## [1] -1  1  3
```

Jika persamaan-persamaan yang terbentuk tersebut divisualisasikan akan terlihat seperti pada Gambar 8.3.

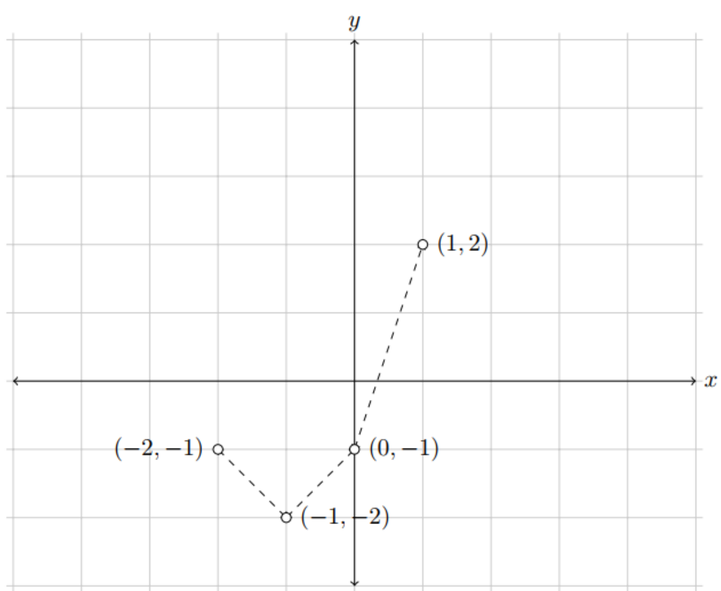

Figure 8.3: Interpolasi linier piecewise empat titik (Sumber:Howard, 2017).

`R` juga menyediakan fungsi untuk melakukan interpolasi linier piecewise. Fungsi `approxfun()` dapat digunakan untuk melakukan interpolasi tersebut. Fungsi `approxfun()` hanya memerlukan dua input yaitu vektor x dan vektor x. Untuk lebih memahaminya, kita akan menggunakan kembali data yang disajikan pada Contoh 8.5 untuk membuat fungsi linier piecewise. Berikut adalah sintaks yang digunakan:

```
f <- approxfun(x, y)

# tentukan nilai y jika x= 0
f(0)
```

```
## [1] -1
```

```
# tentukan nilai y jika x = 0.5
f(0.5)
```

```
## [1] 0.5
```

8.2.2 Interpolasi Spline Kubik

Jika menggunakan interpolasi polinomial berderajat satu (sebuah garis) lebih dari beberapa interval merupakan peningkatan dari satu baris interpolasi, dan jika menggunakan polinomial berderajat tinggi juga merupakan peningkatan dari satu garis interpolasi tunggal, maka dapat disimpulkan bahwa penggunaan polinomial berderajat tinggi pada selang beberapa interval juga akan menjadi peningkatan dalam proses interpolasi. Dalam beberapa kasus, hal tersebut benar, tetapi kita masih menghadapi sudut tajam di mana masing-masing kurva interpolasi tergabung (interpolasi linier piecewise). Sudut tajam ini mencegah diferensiasi dan pada prakteknya tidak dapat digunakan untuk memodelkan beberapa fungsi di dunia nyata, seperti *roller coaster span*.

Interpolasi spline kubik memecahkan masalah ini. Interpolasi ini akan memberikan kurva tergabung yang halus. Hal tersebut juga membuat spline terintegrasi. Karena setiap bagian individu diwakili oleh kurva kubik (polinomial derajat 3), maka masing-masing bagian individu juga dapat dianalisis sebagai kurva kubik. Dengan asumsi ada n titik data untuk interpolasi, kita akan mendefinisikan S_i sebagai fungsi polinomial kubik yang mewakili kurva pada domain $[x_i; x_{i+1}]$. Kemudian untuk n titik observasi, ada $n-1$ interpolasi polinomial kubik.

Bentuk umum seri polinomial dituliskan pada Persamaan (8.9).

$$S_i = d_i \left(x - x_i\right)^3 + c_i \left(x - x_i\right)^2 + b_i \left(x - x_i\right) + a_i \tag{8.9}$$

Ini mengarah ke 4 nilai yang tidak diketahui tidak diketahui, yaitu: a_i, b_i, c_i, dan d_i pada setiap Persamaan (8.9). Oleh karena itu, ada $4\left(n-1\right) = 4n - 4$ nilai yang tidak diketahui. Karena kita ingin spline membentuk garis kontinu dan dapat didiferensiasi, ada satu set persamaan yang menentukan spline kubik:

$$S_i\left(x_i\right) = y_i, \qquad i = 1, \dots\dots, n-1 \tag{8.10}$$

$$S_i\left(x_{i+1}\right)=y_{i+1}, \qquad i=1,\ldots\ldots,n-1 \tag{8.11}$$

$$S_i'\left(x_{i+1}\right)=S_{i+1}'\left(x_i\right), \qquad i=1,\ldots\ldots,n-2 \tag{8.12}$$

$$S_i''\left(x_{i+1}\right)=S_{i+1}''\left(x_i\right), \qquad i=1,\ldots\ldots,n-2 \tag{8.13}$$

Persamaan (8.10) dan (8.11) sudah cukup jelas. Persyaratan ini memastikan bahwa jika kita mengevaluasi spline di salah satu node internal, hasil yang akan kita peroleh merupakan jawaban yang telah ditentukan, yaitu spline yang dievaluasi pada x_i untuk beberapa i adalah y_i, dan setiap komponen spline bergabung dengan rapi. Persamaan (8.12) memastikan bahwa kita memiliki turunan pertama yang berkelanjutan di setiap simpul internal. Ini mencegah terbentuknya sudut tajam pada tiap node. Persamaan (8.13) memastikan turunan kedua juga kontinu, dimana kondisi ini menguntungkan karena itu berarti turunan pertama itu sendiri dapat didiferensiasi juga.

Kondisi ini menyebabkan ada $4n-6$ kondisi yang harus kita penuhi. Jika $4n-4$ kita yang tidak diketahui dipecahkan sebagai sebuah matriks, dan akhirnya matriks tersebut akan terpecahkan, matriks akan menjadi kurang ditentukan. Kita dapat menyelesaikan kondisi tersebut dengan memasukkan dua ketentuan tambahan. Dengan splines kubik, secara normal adalah menentukan akhir di kedua ujung untuk mencapai dua kondisi tambahan. Untuk contoh ini, dua kondisi yang akan kita tambahkan adalah $S_i''\left(x_i\right)=0$ dan $S_i''\left(x_n\right)=0$. Kedua kondisi ini memastikan bahwa pada titik akhir, turunan pertamanya linier dan oleh karena itu fungsi spline berlanjut ke arah yang sudah berjalan. Interpolasi spline kubik ini disebut juga sebagai "natural spline".

Awalnya, kita dapat melihat bahwa setiap polinomial kubik S_i digeser ke kanan oleh unit x_i atau bergeser ke kiri jika x_i negatif. Pada x_i nilai fungsi adalah a_i yang berarti $a_i=y_i$ untuk setiap nilai i. Menyelesaikan sisa koefisien yang ada akan lebih kompleks, tetapi sekarang $3n$ tidak diketahui dengan $3n$ kondisi. Derivasi penuh tersedia dari berbagai sumber, tetapi secara garis besar dari Persamaan (8.12) kita mendapatkan $S_i'\left(x_{i+1}\right)=S_{i+1}'$, kemudian $S_{i+1}'-S_i'\left(x\right)=0$, dan kita dapat mensubstitusi Persamaan (8.9) ke dalam kedua komponen, pemecahan untuk d_i dalam hal ini x_i, y_i, dan c_i. Proses yang sama dapat direplikasi dengan Persamaan (8.13) dan b_i. Hasilnya adalah matriks tridiagonal, seperti yang ada pada Chapter 6.3.3, yang dapat dipecahkan untuk menemukan koefisien. Terdapat sebuah matriks, A, sedemikian rupa sehingga,

$$AC=V \tag{8.14}$$

dimana A merupakan matriks tridiagonal. Pada matriks tridiagonal ini $\left(u_1,u_2,\ldots,u_{n-1}\right)$ merupakan vektor U, dan vektor M, L, dan V ditentukan dengan cara serupa. Matriks ini sedemikian rupa,

$$u_i = l_i = x_{i+1} - x_i \tag{8.15}$$

kecuali pada $u_0 = l_n$. Lebih jauh, diagonal utama D ditentukan menggunakan Persamaan (8.16).

$$m_i = 2\left(x_{i+1} - x_i + x_i - x_{i-1}\right) \tag{8.16}$$

kecuali pada $d_0 = d_n = 1$. Akhirnya vektor V ditentukan dengan Persamaan (8.17).

$$v_i = 3\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right) \tag{8.17}$$

kecuali pada $v_0 = v_n = 0$. Sehingga,

$$\begin{bmatrix} m_1 & u_1 & 0 & 0 & 0 \\ l_1 & m_2 & u_2 & 0 & 0 \\ 0 & l_2 & \ddots & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & u_{n-1} \\ 0 & 0 & 0 & l_{n-1} & m_n \end{bmatrix} C = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ v_n \end{bmatrix} \tag{8.18}$$

Penyelesaian Persamaan (8.18) akan menghasilkan vektor koefisien c, sehingga koefisien b dapat dihitung menggunakan Persamaan (8.19).

$$b_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{3}\left(2c_i + c_{i+1}\right) \tag{8.19}$$

Dengan menggunakan vektor C yang sudah diketahui, koefisien d dapat dihitung menggunakan Persamaan (8.20).

$$d_i = \frac{c_{i+1} - c_i}{3\left(x_{i+1} - x_i\right)} \tag{8.20}$$

Algoritma Interpolasi Spline Kubik

1. Tentukan set titik berpasangan (x, y).
2. Tentukan koefisien a_i menggunakan Persamaan (8.10), dimana $a_i = y_i$.
3. Hitung elemen diagonal bawah (l_i) dan elemen diagonal atas (u_i) matriks tridiagonal menggunakan Persamaan (8.15), dimana $u_0 = l_n = 0$.
4. Hitung elemen diagonal utama (m_i) menggunakan Persamaan (8.16), dimana $d_0 = d_n = 1$.

5. Hitung elemen vektor V menggunakan Persamaan (8.17), dimana $v_0 = v_n = 0$.
6. Susunlah elemen l_i, u_i, dan m_i menjadi matriks tridiagonal A.
7. Deifinisikan persamaan linier seperti pada Persamaan (8.14)
8. Selesaikan sistem persamaan linier pada Persamaan (8.14) sehingga diperoleh vektor C yang merupakan kumpulan koefisien c.
9. Hitung koefisien b_i menggunakan Persamaan (8.19)
10. Hitung koefisien d_i menggunakan Persamaan (8.20).
11. Bentuk seri persamaan polinomial menggunakan elemen koefisien a, b, c, dan d yang telah dihitung.
12. Untuk melakukan ekstrapolasi dengan titik observasi diluar rentang titik diketahui, gunakan persamaan polinomial yang berada pada bagian ujung terdekat dengan nilai x yang hendak dicari nilai y-nya.

Berdasarkan algoritma tersebut, kita dapat menyusun suatu fungsi pada `R` untuk mencari seri persamaan polinomial derajat tiga. Fungsi tersebut adalah sebagai berikut:

```
cubic_spline <- function(x, y){
  n <- length(x)
  d_vec <- b_vec <- a_vec <- rep(0, n-1)
  vec <- rep(0, n)
  delta_x <- delta_y <- rep(0, n-1)

  ## Menghitung nilai selisih dan vektor A
  for(i in 1:(n-1)){
    a_vec[i] <- y[i]
    delta_x[i] <- x[i+1] - x[i]
    delta_y[i] <- y[i+1] - y[i]
  }

  ## Menyusun matriks tridiagona
  Au <- c(0, delta_x[2:(n-1)])
  Am <- c(1, 2*(delta_x[1:(n-2)]+delta_x[2:(n-1)]), 1)
  Al <- c(delta_x[1:(n-2)], 0)

  vec[0] <- vec[n] <- 0
  for(i in 2:(n-1))
    vec[i] <- 3 * (delta_y[i]/delta_x[i] -
                   delta_y[i-1]/delta_x[i-1])

  ## penyelesaian tridiagonal matriks
  nm <- length(Am)
```

```
  Al <- c(NA , Al)

  ### forward sweep
  Au[1] <- Au[1] / Am[1]
  vec[1] <- vec[1] / Am[1]
  for(i in 2:(n - 1)){
      Au[i] <- Au[i] / (Am[i] - Al[i] * Au[i - 1])
      vec[i] <- (vec[i] - Al[i] * vec[i - 1]) /
                (Am[i] - Al[i] * Au[i - 1])
  }
  vec[n] <- (vec[n] - Al[n] * vec[n - 1])/
            (Am[n] - Al[n] * Au[n - 1])

  ### backward sweep
  c_vec <- rep.int (0, n)
  c_vec[n] <- vec[n]
  for(i in (n - 1) :1)
      c_vec[i] <- vec[i] - Au[i] * c_vec[i + 1]

 ## Hitung vektor B dan D dari vektor C
 for(i in 1:(n-1)){
   b_vec[i] <- (delta_y[i]/delta_x[i])-
               (delta_x[i]/3)*(2*c_vec[i]+c_vec[i+1])
   d_vec[i] <- (c_vec[i+1]-c_vec[i]) / (3*delta_x[i])
 }

 return(list(a = a_vec, b = b_vec, c = c_vec[-n], d = d_vec))
}
```

Contoh 8.8. Tentukan persamaan-persamaan spline kubik yang dihasilkan dari titik observasi yang ditampilkan pada Contoh 8.5?

Jawab:

Untuk menentukan persamaan-persamaan linier yang menghubungkan setiap titik, kita akan menggunakan fungsi `cubic_spline()` yang telah kita buat. Berikut adalah sintaks yang digunakan:

```
x <- c(-2, -1, 0, 1)
y <- c(-1, -2, -1, 2)

cubic_spline(x, y)
```

```
## $a
## [1] -1 -2 -1
```

```
##
## $b
## [1] -1.4 -0.2  2.2
##
## $c
## [1] 0.0 1.2 1.2
##
## $d
## [1]  0.4  0.0 -0.4
```

Sebagai contoh untuk domain $[0, 1]$, persamaan spline kubiknya adalah $-0,4x^3 + 1,2x^2 + 2,2x - 1$. Jika persamaan-persamaan yang terbentuk tersebut divisualisasikan akan terlihat seperti pada Gambar 8.4.

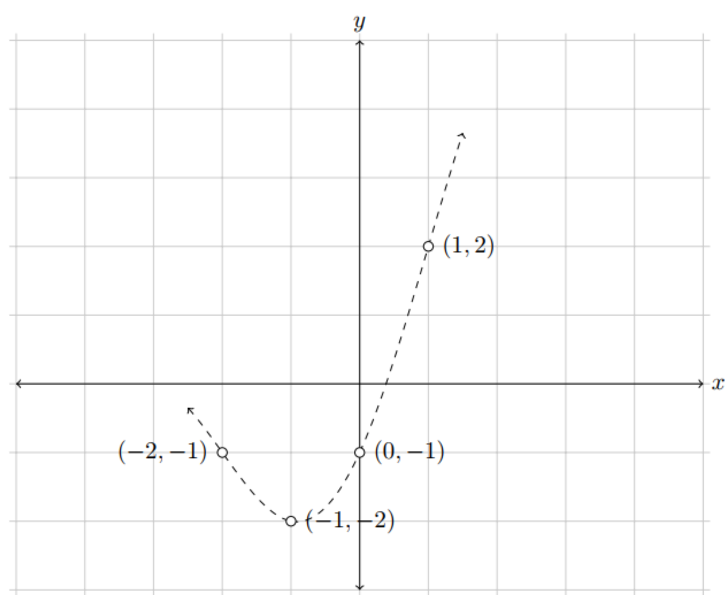

Figure 8.4: Interpolasi spline kubik empat titik (Sumber:Howard, 2017).

Pada `R` juga terdapat fungsi `splinefun()` untuk melakukan interpolasi spline. Format fungsi tersebut adalah sebagai berikut:

```
splinefun(x, y = NULL,
          method = c("fmm", "periodic", "natural", "monoH.FC", "hyman"),
          ties = mean)
```

Catatan:

- `x,y` : vektor titik yang akan dilakukan interpolasi

- `method` : spesifikasi jenis spline yang digunakan. Nilai yang mungkin antara lain: `"fmm"`, `"natural"`, `"periodic"`, `"monoH.FC"` dan `"hyman"`.
- `ties` : metode untuk menangani nilai imbang pada titik yang akan diinterpolasi.

Untuk melakukan interpolasi spline kubik, metode yang digunakan adalah `"natural"`. Berikut adalah contoh interpolasi menggunakan kembali data pada Contoh 8.5:

```
spline <- splinefun(x, y, method="natural")

# uji dengan nilai x yang sama
spline(x)
```

```
## [1] -1 -2 -1  2
```

FUngsi `splinefun()` menghasilkan nilai yang sama persis dengan proses interpolasi yang telah kita lakukan.

8.3 Studi Kasus

Pada studi kasus kali ini, kita akan membahas teknik mengisi nilai yang hilang pada data runtun waktu. Terdapat banyak teknik untuk yang dapat digunakan untuk mengisi data yang hilang. Salah satu teknik yang dapat digunakan adalah dengan melakukan interpolasi.

8.3.1 Interpolasi Data Runtun Waktu

Pengisian data hilang pada data runtun waktu (*time series*) dapat dilakukan dengan berbagai cara sesuai dengan situasi yang dihadapi. Pengisian dapat menggunakan nilai rata-rata jika data memiliki pola *white noise*, observasi terakhir atau observasi dimasa mendatang, dan interpolasi linier.

Data yang digunakan pada contoh kasus kali ini adalah data `airquality`. Dataset tersebut merupakan data kualitas udara bulan Mei sampai September 1973 yang ada di New York. Berikut adalah ringkasan data `airquality` tersebut:

```
summary(airquality)
```

```
##      Ozone          Solar.R         Wind
##  Min.   :  1.0   Min.   :  7   Min.   : 1.70
##  1st Qu.: 18.0   1st Qu.:116   1st Qu.: 7.40
##  Median : 31.5   Median :205   Median : 9.70
##  Mean   : 42.1   Mean   :186   Mean   : 9.96
##  3rd Qu.: 63.2   3rd Qu.:259   3rd Qu.:11.50
##  Max.   :168.0   Max.   :334   Max.   :20.70
##  NA's   :37      NA's   :7
##       Temp          Month           Day
##  Min.   :56.0   Min.   :5.00   Min.   : 1.0
##  1st Qu.:72.0   1st Qu.:6.00   1st Qu.: 8.0
##  Median :79.0   Median :7.00   Median :16.0
##  Mean   :77.9   Mean   :6.99   Mean   :15.8
##  3rd Qu.:85.0   3rd Qu.:8.00   3rd Qu.:23.0
##  Max.   :97.0   Max.   :9.00   Max.   :31.0
##
```

```
str(airquality)
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

```
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

Pada contoh kasus kali ini kita akan mencoba melakukan pengisian data hilang pada data `Solar.R` pada dataset `airquality`. Langkah pertama yang perlu dilakukan adalah membuat objek data runtun waktu pada data tersebut.

```
Solar <- ts(airquality[,"Solar.R"])
```

Visualisasi data tersebut disajikan pada Gambar 8.5.

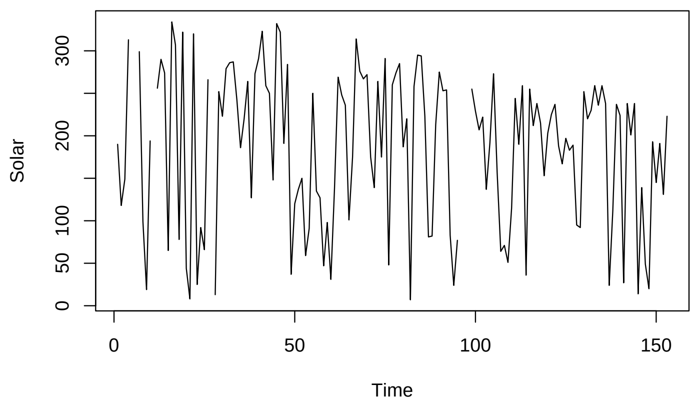

Figure 8.5: Visualisasi variabel radiasi matahari pada dataset airquality.

Berdasarkan visualisasi tersebut terdapat garis yang terputus yang menunjukkan data yang hilang. Agar garis tersebut dapat tersambung, kita perlu melakukan pengisian nilai yang hilang pada data tersebut. Berikut adalah sintaks yang digunakan untuk melakukan pengisian nilai hilang tersebut:

```
library(xts)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
# metode nilai rata-rata
Solar_mean <- na.fill(Solar, fill=mean(Solar, na.rm=TRUE))

# metode last observation carried forward
Solar_locf <- na.locf(Solar)

# metode next observation caried backward
Solar_nocb <- na.locf(Solar, fromLast = TRUE)
```

```r
# metode interpolasi linier
Solar_linterp <- na.approx(Solar)
```

Berikut adalah visualisasi menggunakan metode nilai rata-rata:

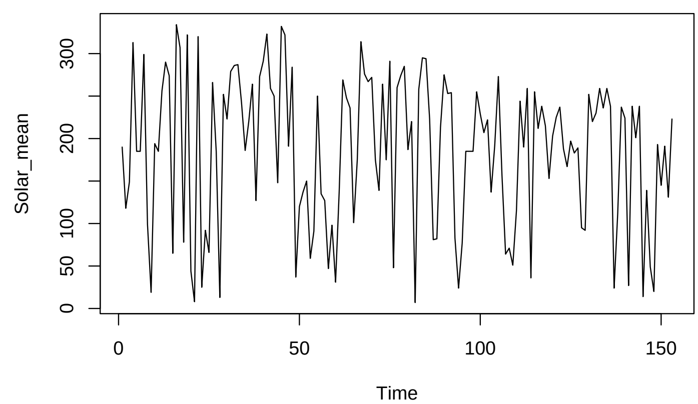

Figure 8.6: Visualisasi data radiasi matahari menggunakan metode nilai rata-rata.

Secara berturut-turut berikut adalah visualisasi dari metode locf, nocb, dan interpolasi linier:

Pemilihan metode interpolasi mana yang sesuai akan berbeda pada setiap situasi dan jenis data yang akan dilakukan interpolasi. Interpolasi data runtun waktu pada bidang lingkungan umumnya menggunakan metode interpolasi nilai rata-rata dan linier dan tidak menutup kemungkinan interpolasi dengan metode lain yang telah dijelaskan pada buku ini dapat pula digunakan.

Pada situasi dimana data membentuk pola *white noises* (pola acak disekitar nilai rata-rata dan memiliki varians yang konstan) seperti yang ditunjukkan variabel `Solar.R`, interpolaasi dengan nilai rata-rata cukup sesuai untuk digunakan untuk mengisi nilai hilang (*missing value*) pada data runtun waktu tersebut.

8.4 Referensi

1. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.

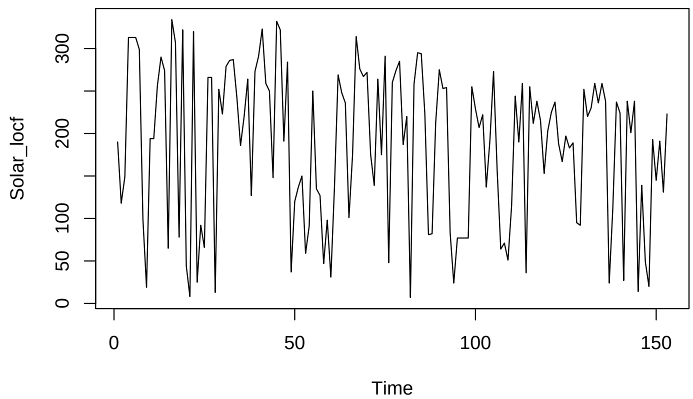

Figure 8.7: Visualisasi data radiasi matahari menggunakan metode locf.

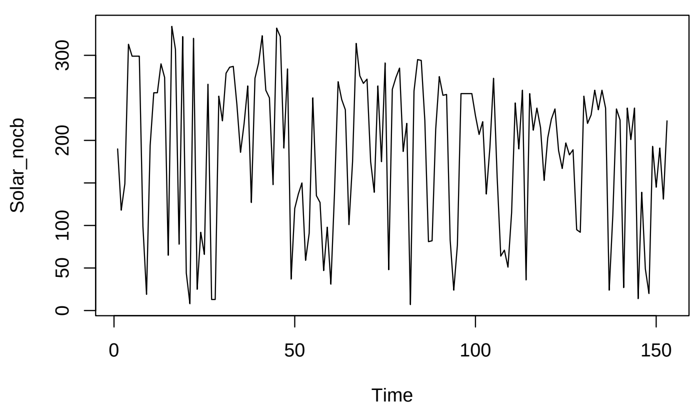

Figure 8.8: Visualisasi data radiasi matahari menggunakan metode nocb.

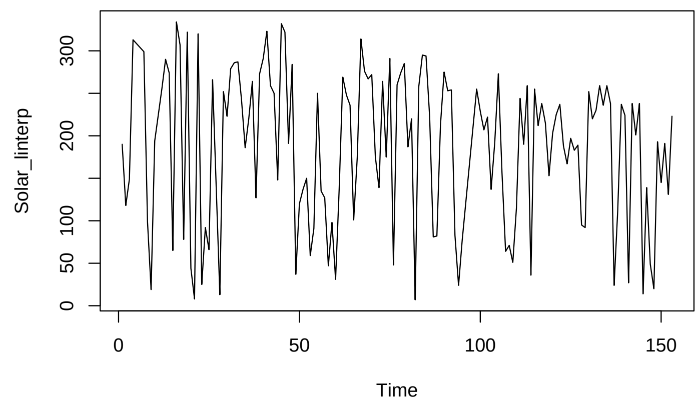

Figure 8.9: Visualisasi data radiasi matahari menggunakan metode interpolasi linier.

2. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
3. Suparno, S. 2008. **Komputasi untuk Sains dan Teknik Edisi II**. Departemen Fisika-FMIPA Universitas Indonesia.

8.5 Latihan

1. Diberikan data titik (3,5), (0,-2), dan (4,1). Tentukan persamaan polinomial untuk melakukan interpolasi pada ketiga titik tersebut!
2. Jika diberikan 13 titik observasi, apakah saudara cenderung akan menggunakan interpolasi polinomial atau spline? jelaskan alasan saudara?
3. Bentuklah kembali fungsi `poly_inter()` dengan menambahkan metode evaluasi polinomial ke dalam fungsi tersebut!
4. Pada Latihan No.1, bentuklah persamaan spline kubik menggunakan titik observasi tersebut!

Chapter 9

Diferensiasi dan Integrasi Numerik

Pada Chapter 9, penulis akan menjabarkan mengenai metode numerik untuk melakukan diferensiasi dan integrasi pada suatu fungsi. Adapun yang akan dibahas pada *Chapter* ini antara lain:

- Metode Beda Hingga
- Metode Integrasi Newton-Cotes
- Metode Integrasi Kudratur Gauss
- Metode Integrasi Adaptif
- Metode Integrasi Romberg
- Metode Integrasi Monte Carlo
- Studi Kasus

9.1 Metode Beda Hingga

Diferensiasi merupakan proses mencari slope suatu garis pada titik yang diberikan. Secara umum proses diferensiasi dinyatakan melalui Persamaan (9.1).

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{9.1}$$

Kita dapat menyatakan secara formal proses diferensiasi sebagai limit Persamaan (9.1) dimana h mendekati nol. Jadi kita ingin membuat nilai h sekecil mungkin untuk memperoleh pendekatan terbaik terhadap nilai turunan suatu fungsi. Kita membatasi nilai h pada sejumlah nilai yang masuk akal untuk mencegah pembagian dengan nilai yang tidak biasa. Kita juga harus memastikan $f(x)$ dan $f(x+h)$ terpisah cukup jauh untuk mencegah *floating point round off error* mempengaruhi proses substraksi.

Terdapat 3 buah metode untuk memperoleh turunan pertama suatu fungsi dengan menggunakan metode numerik, yaitu: metode selisih maju, metode selisih mundur, dan metode selisih tengah. Error pada ketiga metode numerik tersebut ditaksir menggunakan deret Taylor. Persamaan (9.2) dan Persamaan (9.3) menunjukkan persamaan untuk memperoleh turunan pertama dan taksiran error menggunakan metode selisih maju dan metode selisih mundur.

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(c) \tag{9.2}$$

$$f'(x) = \frac{f(x) - f(x-h)}{h} - \frac{h}{2} f''(c) \tag{9.3}$$

Metode nilai tengah menggunakan ukuran langkah h dua kali dibandingkan dengan 2 metode lainnya. Error yang dihasilkan juga berbeda dengan kedua metode sebelumnya, dimana error dihasilkan dari pemotongan turunan ketiga pada deret Taylor. Secara umum metode selisih tengah memiliki akurasi yang lebih baik dibandingkan kedua metode sebelumnya karena metode ini mempertimbangkan dua sisi untuk memeriksa nilai x. Persamaan (9.4) merupakan persamaan untuk memperoleh nilai turunan pertama suatu fungsi dan estimasi error menggunakan deret Taylor.

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(c) \tag{9.4}$$

Bagaimana menentukan h? beberapa literatur menggunakan pendekatan *machine error* ϵ berdasarkan program yang digunakan untuk melakukan proses perhitungan. Metode selisih maju dan selisih mundur menggunakan pendekatan yang ditunjukkan pada Persamaan (9.5).

$$h^* = x\sqrt{\epsilon} \tag{9.5}$$

Untuk metode selisih tengah pendekatan nilai h menggunakan Persamaan (9.6).

$$h^* = x\sqrt[3]{\epsilon} \tag{9.6}$$

Kita dapat menggunakan Persamaan (9.2) sampai Persamaan (9.4) untuk membentuk sebuah program yang digunakan untuk menghitung turunan pertama suatu fungsi. Sintaks yang digunakan adalah sebagai berikut:

```
findiff <- function(f, x, h, method=NULL){
  if(is.null(method)){
    warning("please select a method")
  }else{
    if(method == "forward"){
      return((f(x+h)-f(x))/h)
    }else if(method=="backward"){
      return((f(x)-f(x-h))/h)
    }else if(method=="central"){
      return((f(x+h)-f(x-h))/(2*h))
    }else{
      warning("you can use method: forward, bacward, or central")
    }
  }
}
```

Contoh 9.1. Hitunglah turunan pertama persamaan berikut menggunakan metode selisih titik tengah pada x =1 dan nilai h=0,05!

$$f(x) = e^{-x} \sin(2x) + 1$$

Jawab:

Untuk menghitung turunan pertama menggunakan metode selisih tengah, kita dapat menggunakan Persamaan (9.4). Berikut adalah proses perhitungannya:

$$f'(1) = \frac{f(1 + 0.05) - f(1 - 0,05)}{2 \times 0.05} = -- 0.6390352$$

Dengan menggunakan fungsi `findiff()`, hasil yang diperoleh adalah sebagai berikut:

```
findiff(function(x)
  exp(-x)*sin(2*x)+1, x=1, h=0.05,
  method="central")
```

```
## [1] -0.639
```

Kita dapat memperkecil nilai h untuk memperoleh akurasi yang lebih baik berdasarkan pendekatan Persamaan (9.6).

```
findiff(function(x){  exp(-x)*sin(2*x)+1}, x=1,
        h=1*.Machine$double.eps^(1/3),
        method="central")
```

```
## [1] -0.6407
```

Penyelesaian persamaan matematik dalam bidang Teknik Lingkungan pada umumnya tidak hanya melibatkan turunan pertama, pada penyelesaian persamaan difusi umumnya menggunakan turunan kedua. Persamaan (9.7) merupakan pendekatan numerik untuk memperoleh nilai turunan kedua suatu persamaan dengan pendekatan deret Taylor.

$$f''\left(x\right)=\frac{f\left(x+h\right)-2f\left(x\right)+f\left(x-h\right)}{h^2}-\frac{h^2}{12}f^{(4)}\left(c\right) \tag{9.7}$$

Fungsi `findiff2()` merupakan fungsi yang digunakan untuk menghitung turunan kedua suatu persamaan yang didasarkan pada Persamaan (9.7).

```
findiff2 <- function(f, x, h){
  return((f(x+h)-2*f(x)+f(x-h))/(h^2))
}
```

Kita dapat menghitung kembali turunan kedua fungsi pada Contoh 9.1 menggunakan fungsi `findiff2()`. Berikut adalah sintaks yang digunakan:

```
findiff2(function(x){
  exp(-x)*sin(2*x)+1
}, x=1, h=0.05)
```

```
## [1] -0.3924
```

9.2 Diferensiasi Menggunakan Fungsi Lainnya di R

Terdapat sejumlah fungsi R yang dapat digunakan untuk menghitung turunan suatu persamaan matematik. Fungsi-fungsi tersebut tersedia dalam sejumlah Paket, baik *base package* maupun yang berasal dari Paket lainnya.

9.2.1 Diferensiasi Metode Titik Pusat Mengggunakan Fungsi`diff()`

Fungsi `diff()` pada Paket *base* dapat digunakan untuk menghitung turunan suatu persamaan menggunakan metode titik pusat. Fungsi ini pada umumnya digunakan untuk menghitung *lag* suatu data runtun waktu. Agar fungsi tersebut dapat digunakan untuk menghitung turunan pertama suatu persamaan, kita dapat menggunakan argumen `lag = 2`. Berikut adalah contoh penerapan fungsi `diff()` untuk memperoleh turunan pertama persamaan matematik pada Contoh 9.1:

```
f <- function(x){exp(-x)*sin(2*x)+1}
x <- 1
h <- x*.Machine$double.eps^(1/3)
xvec <- seq(x-h, x+h, h)

# turunan pertama
diff(f(xvec), lag=2)/(2*h)
```

```
## [1] -0.6407
```

9.2.2 Diferensiasi Menggunakan Paket `numDeriv`

Paket standar yang sering digunakan untuk melakukan taksiran numerik turunan suatu fungsi adalah Paket `numDeriv`. Pada Paket tersebut terdapat fungsi `grad()` yang digunakan untuk menaksir turunan pertama suatu persamaan. Format fungsi tersebut adalah sebagai berikut:

```
grad(func, x, method="Richardson", method.args=list(), ...)
```

> **Catatan:**
>
> - **func**: Fungsi persamaan matematik yang akan dicari turunannya.
> - **x**: Lokasi atau titik yang akan dicari gradiennya
> - **method**: Metode estimasi yang digunakan. Metode yang dapat digunakan antara lain:
> - "simple": metode selisih maju dengan $h = 10^{-4}$
> - "Richardson": metode interpolasi Richardson
> - "complex": *complex-step derivative approach* dan dapat digunakan untuk persamaan *complex-differentiable*.
> - **method.args**: argumen tambahan yang digunakan bersama dengan argumen **method**. Jika metode yang digunakan adalah "simple", nilai h dapat dispesifikasikan pada **method.args** jika diinginkan nilai h lainnya.

Berikut adalah contoh penerapan fungsi `grad()` untuk memperoleh turunan pertama persamaan matematik pada Contoh 9.1:

```
numDeriv::grad(function(x){exp(-x)*sin(2*x)+1},
     x=1, method = "simple",
     method.args = list(eps=1*sqrt(.Machine$double.eps)))
```

```
## [1] -0.6407
```

9.2.3 DIferensiasi Menggunakan Paket `pracma`

Terdapat sejumlah fungsi pada Paket `pracma` yang dapat digunakan untuk melakukan diferensiasi suatu persamaan matematik. FUngsi-fungsi yang dapat digunakan untuk melakukan diferensiasi sederhana antara lain: `fderiv()`, `numderiv()`, `numdiff()`, dan `grad()`.

Fungsi `fderiv()` dapat digunakan untuk melakukan diferensiasi orde pertama sampai dengan orde tinggi. Perlu dicatat bahwa diferensiasi cenderung kurang akurat jika orde diferensiasi semakin tinggi. Format yang digunakan untuk melakukan diferensiasi menggunakan fungsi `fderiv()` adalah sebagai berikut:

```
fderiv(f, x, n = 1, h = 0,
        method = c("central", "forward", "backward"),
       ...)
```

> **Catatan:**
>
> - **f**: Fungsi persamaan matematik yang akan dicari turunannya.
> - **x**: Lokasi atau titik yang akan dicari gradiennya
> - **n**: Orde diferensiasi yang digunakan. Orde diferensiasi yang dapat digunakan adalah 1 sampai 8.
> - **method**: Metode estimasi yang digunakan. Metode yang dapat digunakan antara lain:
> - "central": metode titik pusat
> - "forward": metode selisih maju
> - "bacward": metode selisih mundur.
> - **...**: argumen tambahan fungsi **f**.

Berikut adalah contoh penerapan fungsi `fderiv()` untuk memperoleh turunan pertama dan kedua persamaan matematik pada Contoh 9.1:

```
library(pracma)
```

```
##
## Attaching package: 'pracma'
```

```
## The following objects are masked from 'package:rootSolve':
##
##     gradient, hessian

## The following objects are masked from 'package:Matrix':
##
##     expm, lu, tril, triu
```

```
# turunan 1
fderiv(function(x){exp(-x)*sin(2*x)+1},
       x = 1, n = 1, h = 1*.Machine$double.eps^(1/3),
       method = "central")
```

```
## [1] -0.6407
```

```
# turunan 2
fderiv(function(x){exp(-x)*sin(2*x)+1},
       x = 1, n = 2, h = 1*.Machine$double.eps^(1/3),
       method = "central")
```

```
## [1] -0.3912
```

Fungsi `numderiv()` menggunakan ekstrapolasi Richardson untuk melakukan taksiran turunan suatu persamaan matematik. Berbeda dengan fungsi lainnya, fungsi `numderiv()` tidak hanya menampilkan hasil diferensiasi, fungsi ini juga menampilkan error absolut, error relatif, dan jumlah iterasi yang berlangsung. Berikut adalah format fungsi yang digunakan:

```
numderiv(f, x0, maxiter = 16, h = 1/2, ...,
         tol = .Machine$double.eps)
```

> **Catatan:**
>
> - **f**: Fungsi persamaan matematik yang akan dicari turunannya.
> - **x0**: Lokasi atau titik yang akan dicari gradiennya
> - **maxiter**: Iterasi maksimum yang digunakan.
> - **h**: *step size* yang digunakan
> - **tol**: toleransi error yang digunakan.

Berikut adalah contoh penerapan fungsi `numderiv()` untuk memperoleh turunan pertama persamaan matematik pada Contoh 9.1:

```
numderiv(function(x){exp(-x)*sin(2*x)+1},
       x0 = 1, h = 1*.Machine$double.eps^(1/3))
```

```
## $df
## [1] -0.6407
##
## $rel.err
## [1] 7.631e-11
##
## $niter
## [1] 2
```

Fungsi `numderiv()` memiliki keterbatasan dalam penggunaannya. Argumen `x0` yang digunakan haruslah angka numerik tunggal. Fungsi `numdiff()` mengatasi keterbatasan tersebut. Fungsi ini dapat menerima input berupa vektor, sehingga dapat digunakan untuk mencari nilai turunan pada sejumlah titik. Selain itu, output fungsi ini lebih sederhana, dimana hanya menampilkan hasil diferensiasinya saja. Berikut adalah format fungsi `numdiff()`:

```
numdiff(f, x, maxiter = 16, h = 1/2, ...,
         tol = .Machine$double.eps)
```

Catatan:

- **f**: Fungsi persamaan matematik yang akan dicari turunannya.
- **x**: Vektor titik yang akan dicari gradiennya
- **maxiter**: Iterasi maksimum yang digunakan.
- **h**: *step size* yang digunakan
- **tol**: toleransi error yang digunakan.

Berikut adalah contoh penerapan fungsi `numdiff()` untuk memperoleh turunan pertama persamaan matematik pada Contoh 9.1:

```
numdiff(function(x){exp(-x)*sin(2*x)+1},
       x = 1:4, h = 1*.Machine$double.eps^(1/3))
```

```
## [1] -0.64070 -0.07450  0.10952 -0.02345
```

Fungsi `grad()` pada Paket `pracma` berbeda dengan yang digunakan pada Paket `numDeriv`. Perbedaan utama fungsi pada kedua Paket tersebut adalah metode estimasi yang digunakan untuk menghitung turunan pertama suatu persamaan matematik. Pada Paket `pracma`, metode yang digunakan adalah metode titik pusat, sedangkan pada Paket `numDeriv` metode yang digunakan adalah metode selisih maju, ekstrapolasi Richardson, dan *complex*. Format fungsi `grad()` pada Paket `pracma` adalah sebagai berikut:

```
grad(f, x0, heps = .Machine$double.eps^(1/3),
     ...)
```

Catatan:

- **f**: Fungsi persamaan matematik yang akan dicari turunannya.
- **x0**: Titik yang akan dicari gradiennya
- **heps**: *step size* yang digunakan
- **...**: Argumen lain yang digunakan pada fungsi **f**.

Berikut adalah contoh penerapan fungsi `grad()` untuk memperoleh turunan pertama persamaan matematik pada Contoh 9.1:

```
grad(function(x){exp(-x)*sin(2*x)+1}, x0 = 1)
```

```
## [1] -0.6407
```

9.3 Metode Integrasi Newton-Cotes

Metode integrasi Newton-Cotes secara umum merupakan metode integrasi yang dilakukan dengan membagi area di bawah kurva suatu fungsi menjadi beberapa panel dengan terlebih dahulu menetapkan batas atas dan batas bawah interval. Integral atau luas area di bawah kurva ditentukan berdasarkan jumlah luas panel yang digunakan untuk mendekati luas area di bawah kurva.

Terdapat beberapa metode yang akan penulis jelaskan pada sub-Chapter ini. Metode-metode tersebut antara lain:

- Metode integral Riemann
- Metode trapezoida
- Metode Simpson 1/3
- Metode Simpson 3/8

9.3.1 Metode Integral Riemann

Metode integral Riemann dilakukan dengan membagi interval di bawah kurva suatu fungsi matematik sebanyak m subinterval sama besar. Pada setiap subinterval dibentuk persegi panjang setinggi kurva pada setiap titik tengah persegi panjang tersebut. Area setiap subinterval diperoleh dengan mengalikan panjang dan lebar masing-masing persegi panjang. Jumlah masing-masing area tersebut digunakan untuk menaksir interval integral suatu fungsi dengan interval tertentu. Fungsi proses integrasi menggunakan metode titik tengah dapat dituliskan pada Persamaan (9.8).

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{m} f\left(i\,\frac{|b-a|}{m} - \frac{|b-a|}{2m}\right)\,\frac{|b-a|}{m} \tag{9.8}$$

dimana b dan a masing-masing merupakan batas atas dan bawah interval kurva yang hendak dihitung integralnya.

Error dari metode ini dapat diestimasi menggunakan Persamaan (9.9).

$$\int_a^b h(x)\,dx = -\frac{(b-a)^3}{24m^2} f^{(2)}(\xi) \tag{9.9}$$

dimana ξ merupakan nilai antara a dan b.

Contoh 9.2. Hitunglah intergral fungsi di bawah ini menggunakan metode integral Reimann dengan interval 0 sampai 1 dan jumlah panel 2 dan 4!

$$\int_0^1 x^2 dx$$

Jawab:

Fungsi pada Contoh 9.2 dapat diselesaikan menggunakan metode analitik. Penyelesaian analitik fungsi tersebut adalah sebagai berikut:

$$\int_0^1 x^2 dx = \left[\frac{x^3}{3}\right]_0^1 = \frac{1^3}{3} - \frac{0^3}{3} = 0,333...$$

Penyelesaian numerik menggunakan metode titik tengah dengan jumlah panel 2 dapat dilakukan dengan menentukan lokasi titik tengah kedua panel. Berdasarkan interval fungsi dapat kita tentukan titik tengah kedua panel berada pada $x = 0,25$ dan $x = 0,75$. Perhitungan dilakukan seperti berikut:

$$\int_0^1 x^2 dx \approx (f(0,25) + f(0,75))\,\frac{1-0}{2} = \frac{0,25^2 + 0,75^2}{2} = 0,3125$$

Untuk meningkatkan akurasi dari nilai yang dihasilkan, jumlah panel dapat ditingkatkan. Untuk jumlah panel 4, titik tengah berada pada $x = \{0,125; 0,375; 0,625; 0,875\}$.

$$\int_0^1 x^2 dx \approx (f(0,125) + f(0,375) + f(0,625) + f(0,875))\,\frac{1-0}{4} = 0,328125$$

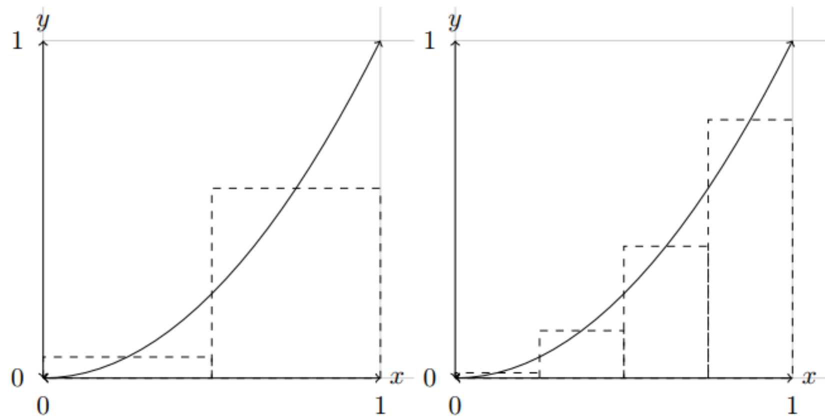

Figure 9.1: Visualisasi integral Riemann dengan 2 panel dan 4 panel (sumber:Howard, 2017).

Visualisasi proses integrasi dengan metode Riemann dapat dilihat pada Gambar 9.1.

Berdasarkan Persamaan (9.8), kita dapat mengembangkan fungsi `R` yang dapat digunakan untuk melakukan perhitungan integral Riemann. Sintaks fungsi tersebut adalah sebagai berikut:

```
riemann <- function(f, a, b, m = 100){
  n_width <- (b-a)/m
  x <- seq(a, b-n_width, length.out = m) + n_width/2
  y <- f(x)

  return(sum(y)*abs(b-a)/m)
}
```

Kita akan menghitung kembali fungsi pada Contoh 9.2 dengan menggunakan jumlah panel 2, 4 dan 100. Berikut adalah sintaks yang digunakan:

```
# m=2
riemann(function(x) x^2, a=0, b=1, m=2)
```

```
## [1] 0.3125
```

```
# m=4
riemann(function(x) x^2, a=0, b=1, m=4)
```

```
## [1] 0.3281
```

```
# m=100
riemann(function(x) x^2, a=0, b=1)
```

```
## [1] 0.3333
```

Berdasarkan teori yang telah dipaparkan sebelumnya, kita ketahui bahwa untuk memperoleh nilai pendekatan integral yang sebenarnya kita dapat meningkatkan jumlah panel yang digunakan. Untuk mengetahui jumlah panel minimum yang diperlukan untuk memperoleh hasil integrasi yang stabil, kita akan melakukan simulasi menggunakan data yang disajikan pada Contoh 9.2 dengan memvariasikan jumlah panel yang akan digunakan. Pada simulasi yang akan dilakukan kita akan coba memvariasikan jumlah panel dari 2 hingga 100. Berikut adalah sintaks yang digunakan:

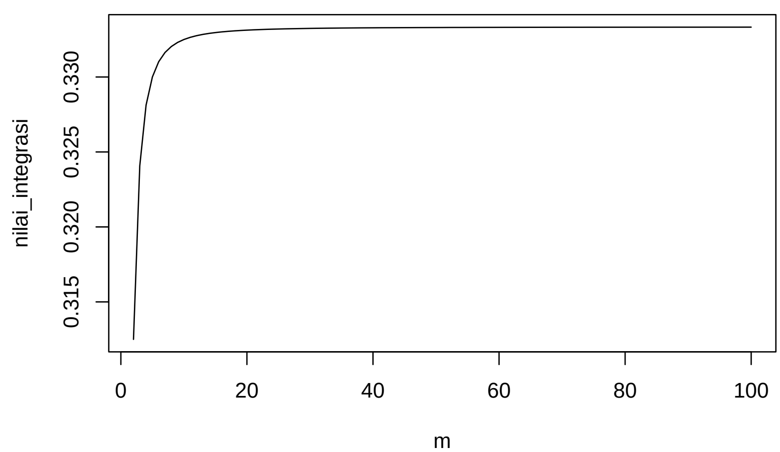

Figure 9.2: Visualisasi simulasi pemilihan jumlah panel minimum metode integrasi Riemann.

Berdasarkan hasil simulasi dapat disimpulkan jumlah panel minimum yang diperlukan untuk memperoleh hasil integrasi yang stabil kira-kira sebesar $m = 40$.

9.3.2 Metode Trapezoida

Pendekatan trapezoida dilakukan dengan melakukan pendekatan area dibawah kurva fungsi $y = f(x)$ dengan subinterval $[x_i, x_{i+1}]$ menggunakan trapesium.

Untuk memahami pendekatan yang digunakan pembaca dapat memperhatikan Gambar 9.3.

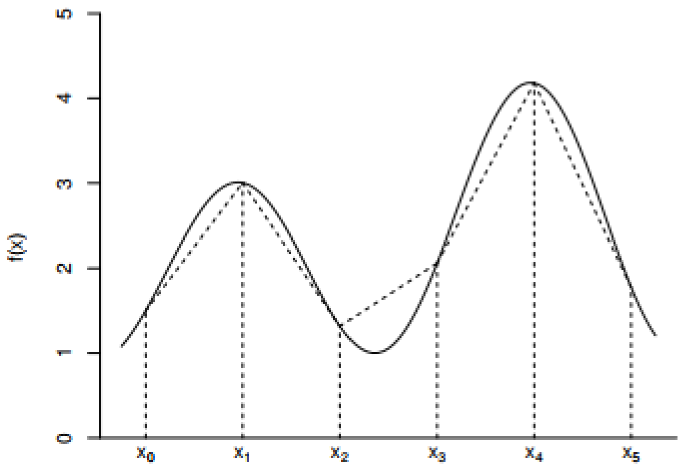

Figure 9.3: Visualisasi integragrasi numerik menggunakan metode tapezoida (sumber: Jones et.al., 2014).

Fungsi proses integrasi menggunakan metode trapezoida dapat dituliskan pada Persamaan (9.10).

$$\int_a^b f(x)\,dx \;=\lim_{m\to\infty}\sum_{i=1}^{m}\frac{(c_{i+1}-c_i)\times(f(c_{i+1})+f(c_i))}{m} \tag{9.10}$$

dimana

$$c_i = a + \frac{(b-a)}{n}i$$

n merupakan nilai subinterval dan m merupakan jumlah panel trapesium yang digunakan.

Error dari metode ini dapat diestimasi menggunakan Persamaan (9.11).

$$\int_a^b h(x)\,dx = -\frac{(b-a)^3}{12m^2}f^{(2)}(\xi) \tag{9.11}$$

dimana ξ merupakan nilai antara a dan b.

Contoh 9.3. Hitung kembali nilai intergrasi persamaan pada Contoh 9.2 menggunakan metode trapezoida dengan jumlah panel m=2!

Jawab:

Penyelesaian numerik menggunakan trapezoida dengan jumlah panel 2 dapat dilakukan dengan menentukan lokasi titik evaluasi. Berdasarkan Gambar 9.4, terdapat 3 batas subinterval yaitu pada a,$\frac{(b-a)}{2}$, dan b. Perhitungan intergral menggunakan ketiga titik evaluasi tersebut adalah sebagai berikut:

$$\int_0^1 x^2 dx \approx \frac{(0,5)\,(0,25+1,25)}{2} = 0,375$$

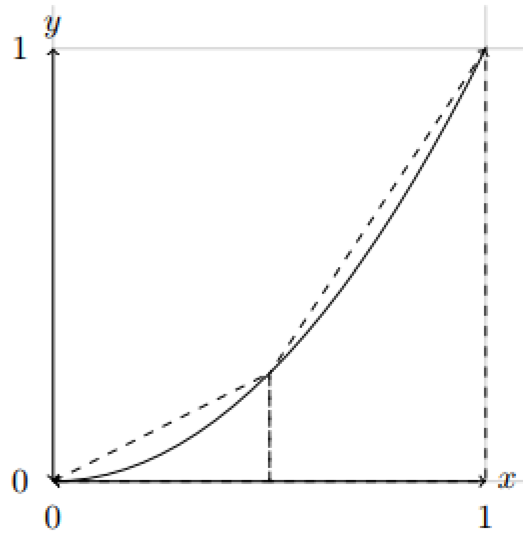

Figure 9.4: Visualisasi integrasi metode trapezoida dengan 2 panel (sumber:Howard, 2017).

Berdasarkan Persamaan (9.10), kita dapat mengembangkan fungsi `R` yang dapat digunakan untuk melakukan perhitungan integral metode trapezoida. Sintaks fungsi tersebut adalah sebagai berikut:

```
trap <- function(f, a, b, m=100){
  x <- seq(a, b, length.out = m+1)
  y <- f(x)

  p_area <- sum((y[2:(m+1)] + y[1:m]))
  p_area <- p_area * abs(b-a)/(2*m)
  return(p_area)
}
```

Kita dapat menghitung kembali intergral persamaan pada Contoh 9.2 menggunakan fungsi `trap()` yang telah dibuat. Berikut adalah sintaks yang digunakan:

```
trap(function(x)x^2, a=0, b=1, m=2)
```

```
## [1] 0.375
```

Untuk mengetahui jumlah panel minimum yang diperlukan untuk memperoleh hasil integrasi yang stabil pada persamaan tersebut, kita akan kembali melakukan simulasi menggunakan variasi jumlah panel yang digunakan. Dalam simulasi variasi jumlah panel yang digunakan adalah 2 sampai 100. Berikut adalah sintaks yang digunakan:

```
## [1] 0.3334
```

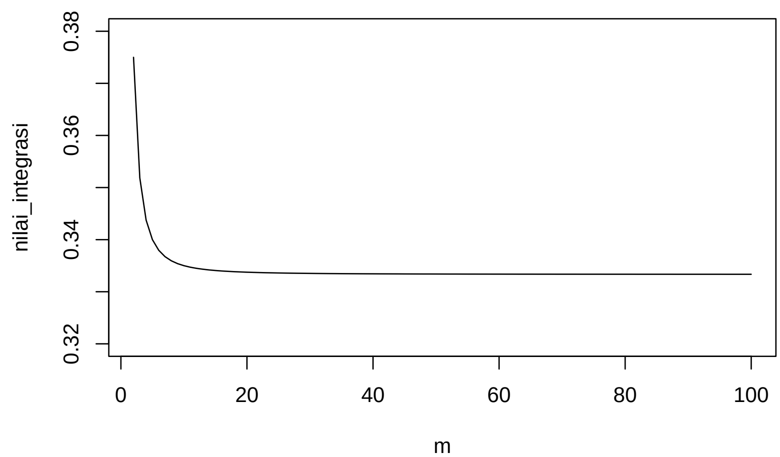

Figure 9.5: Visualisasi simulasi pemilihan jumlah panel minimum metode integrasi trapezoida.

Berdasarkan hasil simulasi diperoleh nilai panel minimum sebesar $m = 20$. Hasil yang diperoleh tersebut menujukkan bahwa metode trapezoida lebih efisien dalam proses komputasi dibandingkan metode Riemann.

9.3.3 Metode Simpson

Metode Simpson membagi subinterval $[a, b]$ menjadi n subinterval, dimana n merupakan bilangan genap. Untuk setiap pasang subinterval, luas area di bawah fungsi $f(x)$ ditaksir menggunakan polinomial berderajat 2.

Misalkan $u < v < w$ merupakan titik sembarang pada suatu fungsi yang akan dicari integralnya yang terpisah sejauh h. Untuk $x \in [u, w]$ kita ingin menaksir $f(x)$ menggunakan parabola yang melalui titik $(u, f(u))$, $(v, f(v))$, dan $(w, f(w))$. Terdapat tepat 1 parabola $p(x)$ yang dapat dibentuk dari ketiga titik koordinat tersebut yang ditunjukkan melalui Persamaan (9.12).

$$p(x) = f(u)\frac{(x-v)(x-w)}{(u-v)(u-w)} + f(v)\frac{(x-u)(x-w)}{(v-u)(v-w)} + f(w)\frac{(x-u)(x-v)}{(w-u)(w-v)} \tag{9.12}$$

Sebagai taksiran luas di bawah kurva $y = f(x)$ digunakan $\int_w^u p(x)\,dx$. Hasil integrasi kurva Persamaan (9.12) disajikan pada Persamaan (9.13).

$$\int_w^u p(x)\,dx = \frac{h}{3}\left(f(u) + 4f(v) + f(w)\right) \tag{9.13}$$

Sekarang asumsikan n merupakan bilangan genap, maka kita perlu menambahkan taksiran untuk subinterval $[x_{2i}, x_{2i+2}]$ untuk memperoleh taksiran S pada integral $\int_a^b f(x)\,dx$ yang disajikan pada Persamaan (9.14).

$$S \approx \frac{h}{3}\left(f_0 + 4\sum_{i=1,3,5...}^{n-1} f_i + 2\sum_{i=2,4,6,...}^{n-2} f_i + f_n\right) \tag{9.14}$$

Persamaan (9.14) disebut sebagai kaidah Simpson 1/3 karena terdapat koefisien 1/3 pada bagian depan persamaan tersebut. Persamaan tersebut juga mudah diingat mengingat pola koefisien persamaan tersebut adalah $1, 4, 2, 4, 2, ..., 2, 4, 1$. Namun penggunaan kaidah 1/3 Simpson mengharuskan jumlah subinterval n genap. Kondisi tersebut jelas berbeda dengan metode trapezoida yang tidak mensyaratkan jumlah selang.

Error dari metode Simpson 1/3 dapat dihitung menggunakan Persamaan (9.14).

$$\int_a^b h(x)\,dx = -\frac{(b-a)^5}{180m^4} f^{(4)}(\xi) \tag{9.15}$$

dimana ξ merupakan nilai antara a dan b.

Algoritma Metode Simpson

1. Tentukan fungsi $f(x)$ dan selang integrasinya $[a, b]$.
2. Tentukan jumlah subinterval n.

3. Hitung nilai selang subinterval h, $h = \frac{b-a}{n}$.
4. Tentukan awal integrasi $x_0 = a$ dan akhir $x_n = b$ dan hitung nilai $f(a)$ dan $f(b)$.
5. Untuk $x = 1, 2, \ldots, n-1$,

- jika ganjil, hitung: $4 \times f(x)$
- jika genap, hitung: $2 \times f(x)$

6. Jumlahkan nilai-nilai taksiran tersebut menggunakan Persamaan (9.14).

Berdasarkan algoritma tersebut, kita dapat membentuk fungsi `simpson()` yang dapat digunakan untuk melakukan integrasi menggunakan metode Simpson 1/3. Berikut adalah sintaks yang digunakan:

```
simpson <- function(f, a, b, m=100){
  h <- (b-a)/m # jarak selang
  x <- a # awal selang
  I <- f(a)+f(b)
  sigma <- 0

  if(m%%2 != 0){
    stop("Jumlah panel harus genap")
  }else{
    for(i in 1:(m-1)){
    x <- x+h
    if(i%%2==0){
      sigma <- sigma + 2*f(x)
    }else{
      sigma <- sigma + 4*f(x)
      }
    }
  }

  return((h/3)*(I+sigma))
}
```

Contoh 9.4. Hitung integral persamaan di bawah ini dengan menggunakan jumlah panel m=8!

$$\int_0^1 \frac{1}{1+x} dx$$

Jawab:

lebar selang h dapat dihitung seperti berikut:

$$h = \frac{1-0}{8} = 0,125$$

Integral persamaan tersebut selanjutnya dapat dihitung menggunakan Persamaan (9.14):

$$S \approx \frac{0,125}{3}\left(f\left(0\right) + 4f\left(0,125\right) + 2f\left(0,25\right) \cdots + f\left(1\right)\right) \approx 0,69412$$

Fungsi `simpson()` juga menghasilkan nilai yang serupa dengan perhitungan manual yang telah dilakukan. Berikut adalah sintaks yang digunakan:

```
simpson(function(x)1/(1+x), a=0, b=1, m=8)
```

```
## [1] 0.6932
```

9.3.4 Metode Simpson 3/8

Jika pada metode Simpson 1/3 digunakan pendekatan polinomial berderajat 2 untuk mencari luas dibawah kurva, pada metode Simpson 3/8 digunakan pendekatan polinomial berderajat 3 untuk memperoleh hasil yang lebih baik. Bentuk umum integrasi yang digunakan disajikan pada Persamaan (9.16).

$$\int_a^b f\left(x\right)dx \approx \frac{3h}{8}\left(f_0 + 3\sum_{i=1;i\neq 3,6,9,\ldots}^{n-1} f_i + 2\sum_{i=3,6,9,\ldots}^{n-3} f_i + f_n\right) \tag{9.16}$$

Error dari metode ini dapat diestimasi menggunakan Persamaan (9.17).

$$\int_a^b h\left(x\right)dx = -\frac{\left(b-a\right)^5}{80m^4}f^{(5)}\left(\xi\right) \tag{9.17}$$

Algoritma Metode Simpson 3/8

1. Tentukan fungsi $f\left(x\right)$ dan selang integrasinya $[a, b]$.
2. Tentukan jumlah subinterval n.
3. Hitung nilai selang subinterval h, $h = \frac{b-a}{n}$.
4. Tentukan awal integrasi $x_0 = a$ dan akhir $x_n = b$ dan hitung nilai $f\left(a\right)$ dan $f\left(b\right)$.

5. Untuk $x = 1, 2, ..., n - 1$,

- jika bukan kelipatan 3, hitung: $3 \times f(x)$
- jika kelipatan 3, hitung: $2 \times f(x)$

6. Jumlahkan nilai-nilai taksiran tersebut menggunakan Persamaan (9.16).

Berdasarkan algoritma tersebut, fungsi `R` dapat disusun untuk melakukan komputasi metode simpson 3/8. Berikut adalah sintaks fungsi tersebut:

```
simpson38 <- function(f, a, b, m=90){
  h <- (b-a)/m # jarak selang
  x <- a # awal selang
  I <- f(a)+f(b)
  sigma <- 0

  if(m%%3 != 0){
    stop("jumlah panel harus kelipatan 3")
  }else{
    for(i in 1:(m-1)){
      x <- x+h
      if(i%%3==0){
      sigma <- sigma + 2*f(x)
      }else{
      sigma <- sigma + 3*f(x)
      }
    }
  }

  return((3*h/8)*(I+sigma))
}
```

Contoh 9.5. Hitung kembali integral persamaan yang disajikan pada Contoh 9.4 menggunakan fungsi simpson38() yang telah dibuat sebelumnya!

Jawab:

Berikut adalah sintaks yang digunakan untuk melakukan proses integrasi Simpson 3/8 menggunakan fungsi `simpson38()`:

```
simpson38(function(x)1/(1+x), a=0, b=1, m=9)
```

```
## [1] 0.6932
```

9.4 Metode Integrasi Newton-Cotes Menggunakan Fungsi Lainnya

Terdapat sejumlah Paket yang menyediakan fungsi yang dapat digunakan untuk melakukan proses integrasi suatu fungsi pada `R`. Paket `pracma` menyediakan fungsi-fungsi seperti `trapz()` dan `cotes()`.

Fungsi `trapz()` merupakan fungsi yang digunakan untuk melakukan integrasi dengan pendekatan trapesium. Penggunaan fungsi tersebut untuk melakukan perhitungan integral tidak dapat dilakukan secara langsung, kita perlu membuat terlebih dahulu seri koordinat `x` dan koordinat `y`. Program selanjutnya akan melakukan interpolasi linier terhadap selang yang saling berdekatan. Luas masing-masing panel selang selanjutnya dihitung dan dijumlahkan untuk memperoleh nilai integral. Format fungsi tersebut adalah sebagai berikut:

```
trapz(x, y)
```

> **Catatan:**
>
> - **x**: vektor sumbu x.
> - **y**: vektor sumbu y.

Untuk lebih memahami penerapannya berikut disajikan contoh untuk mencari integral persamaan pada Contoh 9.2:

```
library(pracma)

f <- function(x)x^2
x <- seq(0, 1, length.out = 101) # membuat subinterval panel sebanyak 100
y <- f(x)

trapz(x, y)
```

```
## [1] 0.3333
```

Fungsi lain yang dapat digunakan untuk menghitung integral suatu fungsi menggunakan metode Newton-Cotes adalah `cotes()`. Pada fungsi tersebut kita perlu menyatakan jumlah subinterval yang digunakan dan jumlah nodes interpolasi yang digunakan. Jumlah nodes akan menentukan fungsi polinomial pendekatan yang digunakan untuk menghitung luas di bawah suatu fungsi. Jika jumlah nodes diatur menjadi dua, maka fungsi pendekatannya berupa garis linier (metode trapezoida). Jika nodes diatur menjadi 3 maka pendekatannya berupa fungsi kuadrat (metode Simpson 1/3). Secara sederhana derajat polinomial n memerlukan $n+1$ nodes. Format fungsi `cotes()` disajikan sebagai berikut:

```
cotes(f, a, b, n, nodes, ...)
```

Catatan:

- **f**: fungsi yang akan dicari integralnya
- **a**: batas atas.
- **b**: batas bawah.
- **n**: jumlah subinterval atau panel.
- **nodes**: jumlah nodes yang digunakan untuk interpolasi fungsi pada tiap subinterval.

Untuk lebih memahami penerapannya berikut adalah contoh perhitungan intergral menggunakan persamaan pada Contoh 9.4.

```
f <- function(x)1/(1+x)
a <- 0; b <- 1

# metode trapezoida
cotes(f, a, b, n=8, nodes=2)
```

```
## [1] 0.6941
```

```
# metode Simpson 1/3
cotes(f, a, b, n=8, nodes=3)
```

```
## [1] 0.6932
```

```
# metode Simpson 3/8
cotes(f, a, b, n=9, nodes=4)
```

```
## [1] 0.6932
```

9.5 Metode Kuadratur Gauss

Metode Newton-Cotes sangat *powerful*, tetapi metode tersebut memiliki dua fitur yang kurang diinginkan. Pertama, kita harus menggunakan evaluasi fungsi sejumlah $n+1$ untuk hasil presisi dan pada polinomial berderajat n. Hal tersebut mungkin tampak seperti rasio yang baik, tetapi dalam praktiknya, jumlah titik evaluasi akan sering ditingkatkan untuk memperoleh akurasi yang lebih tinggi, namun hasil yang diperoleh juga tidak presisi. Sebagai contoh pembaca dapat melakukan simulasi dengan memvariasikan jumlah penel yang digunakan untuk memperoleh nilai integral sebuah fungsi. Jika kita plotkan hasil yang kita peroleh, grafik yang muncul berupa garis yang berosilasi khusunya pada penggunaan polinomial berderajat tinggi yang menunjukkan hasil yang diperoleh menjadi kurang presisi.

Kelemahan kedua, metode Newton-Cotes memerlukan fungsi terintegrasi untuk dievaluasi pada node yang berjarak sama. Ini benar terlepas dari fungsi yang digunakan. Setiap panel membutuhkan node dengan jarak yang sama di dalamnya. Ini bisa menjadi masalah dengan fungsi periodik, di mana diskontinuitas periodik dapat secara kebetulan mendarat di titik evaluasi. Jika panel Newton-Cotes dapat menyebabkan masalah, kita dapat menggunakan integrasi Gaussian untuk menyelesaikan integral.

Secara umum integrasi Gauss berusaha memperoleh pendekatan luas dibawah kurva fungsi dengan memecah fungsi tersebut menjadi faktor bobot c dan $f(x)$ yang merupakan polinomial pendekatannya. Integral diperoleh melalui hasil kali dari bobot dan fungsi polinomial. Jumlah bobot dan fungsi yang digunakan bergantung pada orde n polinomial yang akan digunakan untuk mengestimasi integral suatu fungsi. Bentuk umum dari kaidah Gauss tersebut ditampilkan pada Persamaan (9.18).

$$\int_{-1}^{1} f(x)\,dt \approx c_1 f(x_1) + c_2 f(x_2) + \cdots + c_n f(x_n) \tag{9.18}$$

dimana c merupakan faktor bobot dan x merupakan titik evaluasi.

Nilai faktor bobot dan nilai masing-masing titik evaluasi disajikan pada Gambar 9.6.

Fungsi yang akan dicari nilai integrannya pada umumnya tidak hanya memiliki daerah batas $[-1, 1]$, sehingga pendekatan kuadratur Gauss tidak dapat digunakan secara langsung pada fungsi yang tidak memiliki batas tersebut. Agar kuadratur Gauss tetap dapat digunakan, fungsi tersebut perlu dilakukan transformasi. Proses transformasi dituliskan pada Persamaan (9.19).

$$\int_{a}^{b} f(x)\,dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{a+b+(b-a)\,t}{2}\right) dt \tag{9.19}$$

Algoritma Kuadratur Gauss-Legendre

1. Tentukan fungsi $f(x)$ dan selang integrasinya $[a, b]$.
2. Lakukan transformasi fungsi tersebut hingga diperoleh fungsi dengan selang $[-1, 1]$ menggunakan Persamaan (9.19).
3. Tentukan orde polinomial n yang akan digunakan.

Metode Gauss-Legendre n-titik $\int_{-1}^{1} f(x)dt \approx c_1 f(x_1) + c_2 f(x_2) + \ldots + c_n f(x_n)$			
n	**Faktor bobot**	**Argumen fungsi**	**Galat pemotongan**
2	$c_1 = 1.000000000$ $c_2 = 1.000000000$	$x_1 = -0.577350269$ $x_2 = 0.577350269$	$\approx f^{(4)}(c)$
3	$c_1 = 0.555555556$ $c_2 = 0.888888889$ $c_3 = 0.555555556$	$x_1 = -0.774596669$ $x_2 = 0$ $x_1 = 0.774596669$	$\approx f^{(6)}(c)$
4	$c_1 = 0.347854845$ $c_2 = 0.652145155$ $c_3 = 0.652145155$ $c_3 = 0.347854845$	$x_1 = -0.861136312$ $x_2 = -0.339981044$ $x_3 = 0.339981044$ $x_4 = 0.861136312$	$\approx f^{(8)}(c)$
5	$c_1 = 0.236926885$ $c_2 = 0.478628670$ $c_3 = 0.568888889$ $c_4 = 0.478628670$ $c_5 = 0.236926885$	$x_1 = -0.906179846$ $x_2 = -0.538469310$ $x_3 = 0$ $x_4 = 0.538469310$ $x_5 = 0.906179846$	$\approx f^{(10)}(c)$
6	$c_1 = 0.171324492$ $c_2 = 0.360761573$ $c_3 = 0.467913935$ $c_4 = 0.467913935$ $c_5 = 0.360761573$ $c_6 = 0.171324492$	$x_1 = -0.932469514$ $x_2 = -0.661209386$ $x_3 = -0.238619186$ $x_4 = 0.238619186$ $x_5 = 0.661209386$ $x_6 = 0.932469514$	$\approx f^{(12)}(c)$

Figure 9.6: Tabulasi faktor bobot, titik evaluasi dan galat pemotongan.

4. Lakukan proses integrasi dengan mengalikan faktor bobot c dengan $f(x_i)$ seperti yang ditujukkan pada Persamaan (9.18).

Kita dapat membangun suatu fungsi pada `R` untuk melakukan integrasi Gauss berdasarkan algoritma tersebut. Sintaks fungsi tersebut adalah sebagai berikut:

```
gauss_legendre <- function(f, n){
    if(n==2){
      c <- c(1,1)
      x <- c(-0.577350269, 0.577350269)
      integral <- 0

      for(i in 1:n){
        integral <- integral + c[i]*f(x[i])
      }
    }else if(n==3){
      c <- c(0.555555556,0.888888889,0.555555556)
      x <- c(-0.774596669,0,0.774596669)
      integral <- 0

      for(i in 1:n){
        integral <- integral + c[i]*f(x[i])
      }
    }else if(n==4){
      c <- c(0.347854845,0.652145155,0.652145155,
             0.347854845)
      x <- c(-0.861136312,-0.339981044,0.339981044,
             0.861136312)
      integral <- 0

      for(i in 1:n){
        integral <- integral + c[i]*f(x[i])
      }
    }else if(n==5){
      c <- c(0.236926885,0.478628670,0.568888889,
              0.478628670,0.236926885)
      x <- c(-0.906179846,-0.538469310,0,
             0.538469310, 0.906179846)
      integral <- 0

      for(i in 1:n){
        integral <- integral + c[i]*f(x[i])
      }
```

```
    }else if(n==6){
      c <- c(0.171324492,0.360761573,0.467913935,
             0.467913935,0.360761573,0.171324492)
      x <- c(-0.932469514,-0.661209386,-0.238619186,
             0.238619186, 0.661209386,0.932469514)
      integral <- 0

      for(i in 1:n){
        integral <- integral + c[i]*f(x[i])
      }
    }else{
      stop("n harus ditentukan")
    }
    return(integral)
}
```

Contoh 9.6. Hitunglah integral fungsi berikut menggunakan metode Gauss-Legendre 2 titik!

$$\int_1^2 \left(x^2+1\right) dx$$

Jawab:

Agar fungsi tersebut dapat dicari nilai integralnya menggunakan metode Gauss-Legendre, fungsi tersebut perlu ditransformasi terlebih dahulu:

$$\int_1^2 \left(x^2+1\right) dx = 0,5 \int_{-1}^{1} \left[(1,5+0,5t)^2+1\right] dt$$

Jadi dalam hal ini

$$f(t) = (1,5+0,5t)^2+1$$

maka

$$\int_1^2 \left(x^2+1\right) dx = 0,5 \left[1 \times f\left(\frac{1}{\sqrt{3}}\right) + 1 \times f\left(\frac{1}{\sqrt{3}}\right)\right] \approx 3,333333333$$

Kita juga dapat menggunakan fungsi `gauss_legendre()` untuk melakukan integrasi Gauss-Legendre pada dua titik. Berikut adalah sintaks yang digunakan:

```
0.5*gauss_legendre(function(x)((1.5+0.5*x)^2)+1,
               n=2)
```

```
## [1] 3.333
```

9.6 Metode Gauss-Legendre Menggunakan Fungsi `legendre.quadrature()`

Fungsi `legendre.quadrature()` dari Paket `gaussquand` dapat dijadikan alternatif untuk menghitung integral suatu fungsi menggunakan metode Gauss-Lagendre. Fortmat fungsi tersebut adalah sebagai berikut:

```
legendre.quadrature(functn, rule, lower=-1, upper=1,
  weighted = TRUE, ...)
```

> **Catatan:**
>
> - **functn**: fungsi yang akan dicari integralnya
> - **rule**: data frame yang terdiri atas orde n aturan kuadratur legendre
> - **lower**: batas atas.
> - **upper**: batas bawah.
> - **weighted**: nilai boolean yang menyatakan apakah bobot fungsi disertakan dalam integran
> - **...**: argumen tambahan functn

Untuk menentukan `rule` pada fungsi `legendre.quadrature()`, diperlukan fungsi lain yang tersedia pada Paket `gaussquad`. Fungsi tersebut adalah `legendre.quadrature.rules()`. Fungsi tersebut akan menampilkan list data frame berdasarkan orde polinomial yang digunakan sebagai taksiran yang terdiri atas faktor bobot dan titik evaluasi. Format fungsi tersebut adalah sebagai berikut:

```
legendre.quadrature.rules(n,normalized=FALSE)
```

> **Catatan:**
>
> - **n**: orde tertinggi polinomial yang akan ditampilkan
> - **normalized**: nilai boolean. Jika bernilai `TRUE`, aturan digunakan untuk polinomial ortogonal.

Untuk memahami penerapan kedua fungsi tersebut berikut disajikan contoh penerapannya:

```
library(gaussquad)
# menampilkan aturan untuk orde 4
legendre.quadrature.rules(4)
```

```
## [[1]]
##   x w
## 1 0 2
##
## [[2]]
##         x w
## 1  0.5774 1
## 2 -0.5774 1
##
## [[3]]
##            x      w
## 1  7.746e-01 0.5556
## 2  7.772e-16 0.8889
## 3 -7.746e-01 0.5556
##
## [[4]]
##         x      w
## 1  0.8611 0.3479
## 2  0.3400 0.6521
## 3 -0.3400 0.6521
## 4 -0.8611 0.3479
```

```
# mencari integral suatu fungsi dengan orde
# gauss-legendre sebesar 4
f <- function(x)x^6
rule <- legendre.quadrature.rules(4)[[4]]

legendre.quadrature(f, rule, lower=-1, upper=1)
```

```
## [1] 0.2857
```

9.7 Metode Integrasi Adaptif

Integrasi adaptif menyediakan pendekatan yang berbeda untuk memperoleh nilai intergral suatu fungsi. Salah satu prinsip utama dari analisis numerik adalah bahwa kita harus berkomitmen pada semacam analisis manusia terhadap suatu masalah sebelum mencoba menyelesaikannya secara algoritmik. Metode analisis numerik umumnya tidak dapat menyelesaikan semua masalah dengan sangat baik. Jadi pengetahuan terhadap masalah yang hendak diselesaikan dapat memungkinkan kita memilih metode numerik yang lebih baik sesuai dengan masalah. Misalnya, dalam konteks integrasi numerik, diskontinuitas pada titik akhir tidak akan cocok untuk solusi Newton-Cotes yang bersifat tertutup.

Tentu saja, akan lebih baik jika kita bisa memprogram komputer untuk mempelajari sesuatu tentang masalah, daripada kita melakukan itu sendiri. Metode adaptif memberikan pendekatan untuk melakukan hal ini. Metode integrasi adaptif memeriksa integral yang mereka operasikan dan mengubah parameter mereka sendiri untuk meningkatkan kualitas integrasi. Algoritma adaptif yang paling sederhana memberikan pendekatan *brute force* untuk peningkatan kualitas dengan memeriksa *error* integrasi. Di sisi lain, jika kita tahu *error*-nya, secara teoritis kita bisa memperbaiki estimasi. Di situlah batas *error* pada algoritma Newton-Cotes dapat membantu.

Jika kita dapat menemukan sesuatu tentang *error* tersebut, kita dapat menggunakan informasi itu untuk memperbaiki estimasi pada proses integrasi. Bayangkan kita sedang mengintegrasikan suatu fungsi, $f(x)$ pada batas $[a, b]$. Jika kita menggunakan metode titik tengah (integral Riemann). Kita telah menetahui *error* maksimum yang mungkin terjadi pada metode tersebut melalui Persamaan (9.9). Dua pengamatan segera menjadi jelas. Terlepas dari apa fungsi $f(x)$ itu, atau turunan keduanya, dua perubahan dapat dilakukan pada integrasi untuk meningkatkan kualitasnya. Pertama, *error* adalah proporsi terhadap kubik dari panjang domain integrasi. Mengurangi panjang meningkatkan kualitas dan memotong panjang menjadi dua memberikan peningkatan kualitas delapan kali lipat. Kedua, *error* berbanding terbalik dengan jumlah panel m yang digunakan . Meningkatkan panel mengurangi *error* dan menggandakan jumlah panel yang disediakan dapat meeningkatkan kualitas empat kali lipat.

Kita dapat merancang algoritma di sekitar pengamatan ini. Pertama, kita dapat memperkirakan nilai integral Q_1, menggunakan aturan titik tengah 1-point. Kedua, kita bisa memperkirakannya lagi menggunakan aturan titik tengah 2-point Q_2. Karena kita telah menggandakan jumlah titik dalam aturan, kita sekarang tahu bahwa perbedaan maksimum antara Q_1 dan Q, nilai sebenarnya dari integral, tidak lebih dari 4 kali lebih besar dari perbedaan maksimum antara Q_2 dan Q. Jika $Q_2 - Q$ kurang dari toleransi tertentu, maka perbedaan antara Q_1 dan Q_2 harus kurang dari tiga kali toleransi yang sama.

Kita peeriksa perbedaan antara dua perkiraan. Jika perbedaannya lebih besar dari toleransi, kita mungkin masih berada dalam toleransi, tetapi kita belum pasti. Jadi proses membagi wilayah integrasi menjadi dua, dan menerapkan integrator adaptif untuk kedua bagian, secara terpisah, menjumlahkan hasilnya akan terus dilakukan.

Algoritma Metode Riemann Adaptif

1. Tentukan fungsi $f(x)$ dan selang integrasinya $[a, b]$.
2. Tentukan jumlah subinterval m.
3. Jika $m = 1$, hitung luas area di bawah kurva dengan pendekatan metode Riemann dengan $m = 2$.
4. Jika $n > 1$,

- Hitung Q_1 dengan pendekatan metode Riemann dan $m = 1$
- Hitung Q_2 dengan pendekatan metode Riemann dan $m = 2$

5. Jika $Q_1 - Q_2 > 3 \times$ nilai toleransi,

- Perkecil m sebanyak 1, $m - 1$
- Perkecil nilai toleransi menjadi setengahnya
- Bagi area integrasi menjadi dua bagian dengan menetapkan c sebagai batas, sehingga terdapat dua batas yaitu: $[a, c]$ dan $[c, b]$.
- Lakukan perhitungan kembali integral pada masing-masing batas tersebut menggunakan metode Riemann dan cek apakah $Q_1 - Q_2 > 3 \times$ nilai toleransi.

6. Jika $Q_1 - Q_2 < 3 \times$ nilai toleransi, luas integral $= Q_2$.

Kita dapat membangun sebuah fungsi integral adaptif menggunakan algoritma tersebut. Sintaks fungsi tersebut adalah sebagai berikut:

```
riemann_adaptint <- function(f, a, b, m=10, tol=1e-8){
  if(m<1){
    stop("m harus >= 1")
  }else if(m==1){
    m <- 2
    n_width <- (b-a)/m
    x <- seq(a, b-n_width, length.out = m) + n_width/2
    y <- f(x)
    area <- sum(y)*abs(b-a)/m
  }else{

    m1 <- 1
    n_width1 <- (b-a)/m1
    x1 <- seq(a, b-n_width1, length.out = m1) + n_width1/2
    y1 <- f(x1)
```

```
    q1 <- sum(y1)*abs(b-a)/m1

    m2 <- 2
    n_width2 <- (b-a)/m2
    x2 <- seq(a, b-n_width2, length.out = m2) + n_width2/2
    y2 <- f(x2)
    q2 <- sum(y2)*abs(b-a)/m2

    if(abs(q1-q2)>3*tol){
      m <- m-1
      tol <- tol/2
      c <- (a+b)/2
      lt <- riemann_adaptint(f, a, c, m=m, tol=tol)
      rt <- riemann_adaptint(f, c, b, m=m, tol=tol)
      area <- lt+rt
    }else{
      area <- q2
    }
  }

  return(area)
}
```

Contoh 9.7. Hitung integral fungsi di bawah ini dengan menggunakan integral Riemann adaptif dengan jumlah panel yang digunakan m=100!

$$\int_{1}^{10} \sin\left(x\right)^{2} + \log\left(x\right) dx$$

Jawab:

Kita akan menghitung integral dari fungsi tersebut menggunakan fungsi `R` yang telah dibuat. Berikut adalah sintaks yang digunakan:

```
riemann_adaptint(function(x)sin(x)^2 + log(x),
                 a=1, b=10, m=100)
```

```
## [1] 18.52
```

9.8 Metode Integral Adaptif Menggunakan Fungsi Lainnya Pada R

Terdapat dua buah fungsi yang hendak penulis kenalkan pada pembaca yang berfungsi untuk melakukan integrasi adaptif pada `R`. Fungsi-fungsi tersebut antara lain: `integrate()` dari Paket `base` dan `integral()` dari Paket `pracma`.

Fungsi `integrate()` merupakan fungsi yang akan melakukan integrasi numerik menggunakan metode kudratur adaptif untuk sebuah variabel dengan selang terbatas (*finite*) maupun tidak terbatas (*infinite*). Format fungsi tersebut secara umum adalah sebagai berikut:

```
integrate(f, lower, upper, ..., subdivisions = 100L,
          rel.tol = .Machine$double.eps^0.25,
          abs.tol = rel.tol)
```

Catatan:

- **f**: fungsi yang akan dicari integralnya
- **lower**: batas bawah.
- **upper**: batas atas.
- **...**: argumen tambahan functn
- **subdivision**: jumlah subinterval atau panel yang akan digunakan.
- **rel.tol**: nilai akurasi relatif yang hendak dicapai
- **abs.tol**: nilai akurasi absolut yang hendak dicapai

Contoh penerapan fungsi `integrate()` adalah sebagai berikut:

```
integrate(function(x)sin(x)^2 + log(x),
          lower=1, upper=10,
          rel.tol = 1e-8)
```

```
## 18.52 with absolute error < 4.1e-10
```

Fungsi lainnya yang dapat digunakan untuk melakukan komputasi integral adaptif adalah fungsi `integral()` dari Paket `pracma`. Terdapat dua buah metode integrasi adaptif yang dapat digunakan pada fungsi tersebut yaitu: Gauss-Konrod dan Simpson. Metode Clenshaw-Curtis yang tersedia masih belum dapat melakukan integrasi adaptif melalui fungsi tersebut. Format umum fungsi `integral()` adalah sebagai berikut:

```
integral(fun, xmin, xmax,
         method = c("Kronrod", "Clenshaw","Simpson"),
         no_intervals = 8, reltol = 1e-8,
         abstol = 0, ...)
```

Catatan:

- **fun**: fungsi yang akan dicari integralnya

- **xmin**: batas bawah.
- **xmax**: batas atas.
- **method**: metode imtegrasi yang digunakan.
- **...**: argumen tambahan fun.
- **no_intervals**: jumlah subinterval atau panel yang akan digunakan.
- **reltol**: nilai akurasi relatif yang hendak dicapai
- **abstol**: nilai akurasi absolut yang hendak dicapai

Berikut merupakan contoh penerapan fungsi `integral()`:

```
pracma::integral(function(x)sin(x)^2 + log(x),
          xmin=1, xmax=10, method="Simpson",
          reltol = 1e-8)
```

```
## [1] 18.52
```

9.9 Metode Integrasi Romberg

Seperti halnya algoritma integrasi adaptif, integrasi Romberg adalah perluasan yang relatif mudah dari keluarga algoritma Newton-Cotes. Keduanya bekerja dengan menggunakan iterasi yang disempurnakan dari beberapa metode Newton-Cotes yang mendasarinya untuk memberikan perkiraan nilai integral yang lebih akurat. Tidak seperti proses komputasi fungsi `riemann_adapint()`, integrasi Romberg bukanlah pendekatan adaptif terhadap integrasi. Hal tersebut berarti metode Romberg tidak mengubah perilakunya sendiri berdasarkan perilaku fungsi yang akan diintegrasikan. Sebaliknya, kita mengeksploitasi perilaku fungsi trapesium pada batas untuk menghasilkan estimasi integral.

Untuk memahami integrasi Romberg, kita harus mulai dengan implementasi rekursif dari aturan trapesium. Jika kita mulai dengan suatu fungsi, $T(f, m)$ di mana T adalah fungsi trapesium, f adalah fungsi yang akan diintegrasikan, dan m adalah jumlah panel untuk diintegrasikan, maka,

$$S(f, m) = \frac{4T(f, m) - T(f, m/2)}{3} \tag{9.20}$$

di mana S adalah aturan Simpson. Kemudian, jika kita mendefinisikan $T(f, 0) = (b - a)(f(b) + f(a)) = 2$, maka fungsi rekursif kita selesai, karena berdasarkan hubungan ini, fraksi yang diberikan dalam Persamaan (9.20) juga merupakan perkiraan untuk integral.

Secara umum,

$$I_{j,k} = \frac{4^k I_{j,k-1} - I_{j-1,k-1}}{4^k - 1} \tag{9.21}$$

di mana $I_{0,0}$ adalah aturan trapesium satu panel dan $I_{j,0}$ adalah aturan trapesium dengan panel 2^j. Dengan menggunakan fungsi-fungsi dasar ini, $I_{j,k}$ dapat ditemukan secara iteratif sebagai matriks segitiga-bawah di mana masing-masing nilai di kolom yang bukan paling kiri adalah fungsi dari nilai di sebelah kiri dan entri di atasnya.

Definisi rekursif ini muncul dari ekstrapolasi Richardson. Ketika diterapkan pada algoritma trapesium, yang konvergen menuju nilai sebenarnya dari integral sebagai m (jumlah panel) meningkat, hubungan dalam Persamaan (9.21) muncul. Penting untuk dipahami bahwa pada batas ketika k mendekati tak terhingga, nilai $I_{j,k}$ adalah nilai sejati integral. Untuk nilai yang lebih kecil dari k, integral Romberg masih hanya perkiraan, meskipun hasil yang diperoleh sangat bagus.

Algoritma Metode Integrasi Romberg

1. Tentukan fungsi $f(x)$ dan selang integrasinya $[a, b]$.
2. Tentukan jumlah subinterval m.
3. Bentuk matrik R dengan ukuran $m \times m$ yang akan menampung hasil perhitungan.
4. Untuk $R_{1,1}$ hitung integral fungsi menggunakan metode trapezoida dengan $m = 1$.
5. Untuk $j = 2, \ldots, m$ dan $k = 1$, hitung integral dengan jumlah panel $m = 2^{j-1}$
6. Untuk $j = 2, \ldots, m$ dan $k = 2, \ldots, m$ hitung nilai perbaikan nilai integrasi menggunakan Persamaan (9.21).
7. Solusi integrasi diperoleh pada $R_{m,m}$.

Berdasarkan algoritma tersebut, kita akan menyusun suatu fungsi pada `R` untuk melakukan proses komputasi integrasi dengan metode Romberg. Berikut adalah sintaks fungsi yang dibuat:

```
romberg <- function(f, a, b, m, tab=FALSE){
  R <- matrix(NA, nrow=m, ncol=m)

  R[1,1] <- trap(f, a, b, m=1)
  for(j in 2:m){
```

```
    R[j,1] <- trap(f, a, b, m=2^(j-1))
    for(k in 2:j){
      k4 <- 4^(k-1)
      R[j,k] <- (k4*R[j,k-1]-R[j-1,k-1])/(k4-1)
    }
  }

  if(tab==TRUE){
    return(R)
  }else{
    return(R[m,m])
  }
}
```

Contoh 9.8. Hitung integral fungsi yang ditampilkan pada Contoh 9.7 dengan m=10!

Jawab:

Kita dapat menggunakan fungsi `romberg()` untuk melakukan proses integrasi menggunakan metode Romberg. Berikut adalah sintaks yang digunakan:

```
# menampilkan matriks proses perhitungan
romberg(function(x)sin(x)^2 + log(x),
        a=1, b=10, m=10, tab=TRUE)
```

```
##        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
##  [1,] 14.88    NA    NA    NA    NA    NA    NA    NA
##  [2,] 17.35 18.18    NA    NA    NA    NA    NA    NA
##  [3,] 18.19 18.47 18.48    NA    NA    NA    NA    NA
##  [4,] 18.43 18.52 18.52 18.52    NA    NA    NA    NA
##  [5,] 18.50 18.52 18.52 18.52 18.52    NA    NA    NA
##  [6,] 18.52 18.52 18.52 18.52 18.52 18.52    NA    NA
##  [7,] 18.52 18.52 18.52 18.52 18.52 18.52 18.52    NA
##  [8,] 18.52 18.52 18.52 18.52 18.52 18.52 18.52 18.52
##  [9,] 18.52 18.52 18.52 18.52 18.52 18.52 18.52 18.52
## [10,] 18.52 18.52 18.52 18.52 18.52 18.52 18.52 18.52
##        [,9] [,10]
##  [1,]    NA    NA
##  [2,]    NA    NA
##  [3,]    NA    NA
##  [4,]    NA    NA
##  [5,]    NA    NA
##  [6,]    NA    NA
##  [7,]    NA    NA
```

```
##  [8,]    NA    NA
##  [9,] 18.52    NA
## [10,] 18.52 18.52
```

Berdasarkan hasil perhitungan nilai integral fungsi tersebut adalah 18.5249.

9.10 Metode Integrasi Romberg Menggunakan Fungsi Lainnya

Fungsi `romberg()` pada Paket `pracma` dapat digunakan untuk melakukan integrasi metode Romberg. Format umum fungsi tersebut adalah sebagai berikut:

```r
romberg(f, a, b, maxit = 25, tol = 1e-12, ...)
```

Catatan:

- **f**: fungsi yang akan dicari integralnya
- **a**: batas bawah.
- **b**: batas atas.
- **...**: argumen tambahan functn
- **maxit**: jumlah iterasi maksimum.
- **tol**: nilai akurasi yang hendak dicapai

Berikut adalah contoh penerapan fungsi `romberg()`:

```r
pracma::romberg(function(x)sin(x)^2 + log(x),
        a=1, b=10)
```

```
## $value
## [1] 18.52
##
## $iter
## [1] 8
##
## $rel.error
## [1] 7.105e-15
```

9.11 Metode Integrasi Monte Carlo

Nama Monte Carlo berasal dari daerah di Monako, yang terkenal karena aktikvitas kasino dan perjudiannya. Jelas, permainan kasino yang baik tergantung pada keacakan, seperti juga metode Monte Carlo. Nama ini menggambarkan pentingnya keacakan dalam proses karena algoritma Monte Carlo menggunakan generator angka acak untuk membedakan input ke suatu fungsi.

Angka acak harus berasal dari domain fungsi yang diharapkan. Selanjutnya, fungsi itu sendiri bersifat deterministik karena untuk diberikan dua input dari domain fungsi x_1 dan x_2. Jika $x_1 = x_2$, maka $f(x_1) = f(x_2)$. Generator angka acak digunakan untuk menghasilkan sejumlah besar input dan fungsinya dijalankan pada setiap input. Akhirnya, hasil yang diperoleh dikumpulkan sesuai dengan model logika yang sesuai dengan analisis yang dilakukan.

Metode Monte Carlo dapat digunakan untuk integrasi numerik dalam jumlah dimensi apa pun yang diberikan. Pendekatan mendasar metode ini adalah menempatkan beberapa titik m secara acak di atas domain untuk diintegrasikan. Jika titik terletak "di bawah" garis fungsi, maka titik tersebut dianggap dalam area integrasi. Jika titiknya "di atas" garis fungsi, maka titik tersebut bukan berada diluar garis integrasi. Area di bawah perkiraan kurva adalah persentase titik di bawah garis.

Beberapa algoritma Monte Carlo yang paling awal digunakan untuk menemukan area di bawah kurva atau untuk memperkirakan nilai π sebuah hobi favorit matematikawan sejak dahulu. Satu pendekatan menciptakan seperempat lingkaran, menggunakan fungsi $f(x) = \sqrt{1 - x^2}$. Melalui domain $[0, 1]$, ini adalah fungsi dan merupakan hasil dari penyelesaian persamaan standar untuk lingkaran, $x^2 + y^2 = r$ untuk y di mana $r = 1$.

Gambar 9.7 menunjukkan plot fungsi ini. Selain itu, 20 titik acak dipilih. Jika titik di bawah kurva dilambangkan dengan lingkaran hitam terisi dan titik-titik kurva dilambangkan dengan titik bulat kosong. Dalam contoh ini, terdapat 15 titik berada di bawah kurva, mengarah ke estimasi area luas area 0,75. Karena kurva mewakili seperempat lingkaran, estimasi untuk π adalah 3. Meningkatkan jumlah tes titik acak meningkatkan ketepatan estimasi dan akurasi.

Bentuk umum metode Monte-Carlo disajikan pada Persamaan (9.22).

$$\int_a^b f(x)\, dx \approx (b - a) \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{9.22}$$

dimana N merupakan jumlah titik yang akan dievaluasi.

Algoritma Metode Monte Carlo

1. Tentukan fungsi $f(x)$ dan selang integrasinya $[a, b]$.
2. Tentukan jumlah titik acak yang akan digunakan N.
3. Lakukan produksi titik acak x dengan selang $[a, b]$ sejumlah N

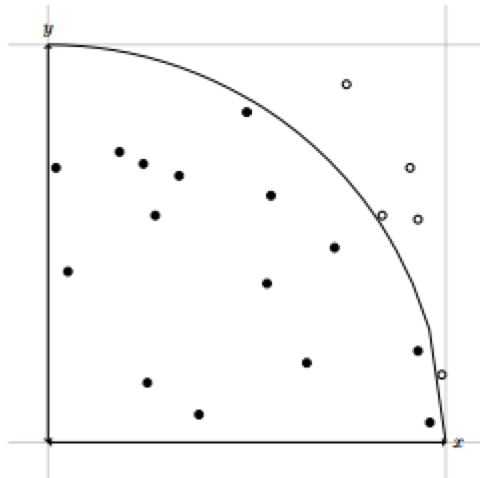

Figure 9.7: Visualisasi metode Monte-Carlo untuk fungsi setengah lingkaran dengan jumlah bilangan acak 20 (sumber: Jones et.al., 2014).

4. Hitung $f(x_i)$
5. Hitung estimasi area menggunakan Persamaan (9.22)

Berdasarkan algoritma tersebut, fungsi R dapat dibangun untuk melakukan integrasi numerik menggunakan metode Monte Carlo. Berikut adalah sintaks yang digunakan:

```
monte_int <- function(f, a, b, m=1e6){
  x <- runif(m, min=a, max=b)

  return((b-a)*sum(f(x))/m)
}
```

Contoh 9.9. Hitung integral fungsi yang ditampilkan pada Contoh 9.7 menggunakan metode Monte-Carlo dengan m=1e6!

Jawab:

Integrasi Monte Carlo menggunakan menggunakan fungsi `monte_int()` disajikan pada sintaks berikut:

```
monte_int(function(x)sin(x)^2 + log(x), a=1, b=10)
```

```
## [1] 18.52
```

Hasil yang diperoleh sedikit berbeda dengan yang dihasilkan oleh metode lainnya. Hal ini disebabkan oleh penggunaan bilangan acak pada proses integrasi. Selain itu, metode ini juga menghasilkan kualitas hasil yang rendah dengan tingkat komputasi yang tinggi dibandingkan dengan metode Newton-Cotes.

Keunggulan metode Monte Carlo dibandingkan metode sebelumnya adalah kemampuan untuk menangani proses integrasi berganda. Berikut adalah bentuk umum proses integrasi bivariat menggunakan metode Monte Carlo:

$$\int\int f\left(x,y\right)dx \approx V\frac{1}{N}\sum_{i=1}^{N} f\left(x_i,y_i\right) \tag{9.23}$$

dimana V merupakan area perpotongan $x-y$ dimana fungsi $f\left(x,y\right)$ diintegrasikan.

Algoritma Metode Monte Carlo Bivariat

1. Tentukan fungsi $f\left(x\right)$ dan domain integrasinya pada masing-masing sumbu x dan y
2. Tentukan jumlah titik acak yang akan digunakan N.
3. Lakukan produksi titik acak x dan y masing-masing domain sumbunya sejumlah N
4. Hitung $V=\left(x_{max}-x_{min}\right)\times\left(y_{max}-y_{max}\right)$
5. Hitung estimasi volume menggunakan Persamaan (9.23)

```
monte_int2 <- function(f, xdom, ydom, m=1000){
  xmin <- min(xdom)
  xmax <- max(xdom)
  ymin <- min(ydom)
  ymax <- max(ydom)

  x <- runif(m, min=xmin, max=xmax)
  y <- runif(m, min=ymin, max=ymax)

  V <- (xmax-xmin)*(ymax-ymin)

  return(V*sum(f(x,y))/m)
}
```

Contoh 9.10. Hitung volume melalui integrasi persamaan berikut menggunakan metode Monte Carlo dengan domain x = [0,1] dan y=[0,1]!

$$\int_0^1 \int_0^1 x^2 y \; dy \; dx$$

Jawab:

Berikut adalah sintaks yang digunakan untuk melakukan integrasi persamaan tersebut menggunakan metode Monte Carlo bivariat:

```
monte_int2(function(x,y)x^2*y,
           xdom=c(0,1), ydom=c(0,1))
```

```
## [1] 0.168
```

Karena metode Monte Carlo tidak deterministik, *error* integrasi Monte Carlo tidak dibatasi dalam pengertian yang telah kita lihat sejauh ini. Namun, kita dapat memperkirakan varians dari estimasi yang dihasilkan, yang berkurang dengan meningkatnya jumlah poin:

$$\text{Var} \frac{1}{N} \sum f\left(\right) = \frac{\sigma^2}{N} \tag{9.24}$$

dimana $\sigma^2 = \text{Var}\; f\left(\right)$. Definisi ini juga digunakan untuk proses integrasi dengan dimensi yang lebih tinggi.

Perlu diketahui pula bahwa metode Monte Carlo hanya dapat digunakan jika nilai terendah dari variabel bebas yang digunakan tidak negatif. Hal ini dilakukan untuk mencegah adanya pengurangan dengan nilai negatif sehingga hasil integrasi jauh lebih besar dari yang seharusnya.

9.12 Studi Kasus

9.12.1 Penerjung Payung

Pada studi kasus kali ini, penulis akan memberikan contoh penerapan integrasi numerik dalam menganalisa jarak jatuh seorang penerjung payung yang melompat dari sebuah pesawat. Kecepatan penerjun payung dapat dituliskan ke dalam sebuah fungsi dari waktu,

$$v\left(t\right) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right) \tag{9.25}$$

dimana v adalah kecepatan penerjun payung dalam m/dt, g adalah percepatan gravitasi sebesar $9,8\ m/dt^2$, m adalah massa penerjun payung sebesar $68,1\ kg$, dan c adalah koefisien tahanan udara sebesar $12,5\ kg/dt$.

Misalkan kita ingin mengetahui seberapa jauh penerjun telah jatuh setelah waktu tertentu t. Karena kecepatan merupakan turunan pertama dari fungsi jarak, maka jarak penerjun dari titik terjun ($t = 0$) adalah:

$$d = \int_0^t v(t)\, dt = \int_0^t \frac{gm}{c} \left(1 - e^{-(c/m)t}\right) dt \tag{9.26}$$

Jika kita ingin mengetahui jarak yang telah ditempuh saat $t = 10$, kita dapat melakukan integrasi pada persamaan tersebut dengan domain $t = [0, 10]$. Persamaan (9.26) dapat dinyatakan menjadi Persamaan (9.27) dengan memasukkan semua komponen yang telah diketahui sebelumnya.

$$d = \int_0^{10} \frac{9,8 \times 68,1}{12,5} \left(1 - e^{-(12,5/68,1)^t}\right) dt \tag{9.27}$$

Kita dapat menyelesaikan Persamaan (9.27) dengan menggunakan metode-metode integrasi numerik yang telah dijabarkan sebelumnya. Berikut adalah sintaks untuk masing-masing metode tersebut:

```
f <- function(x)((9.8*68.1)/12.5)*(1-exp(-(12.5/68.1)*x))
a <- 0; b <- 10
```

Newton-Cotes

```
# Metode Riemann
riemann(f, a, b)
```

```
## [1] 289.4
```

```
# Metode Trapezoida
trap(f, a, b)
```

```
## [1] 289.4
```

```
# Metode Simpson 1/3
simpson(f, a, b)
```

```
## [1] 289.4
```

```
# Metode Simpson 3/8
simpson38(f, a, b)
```

```
## [1] 289.4
```

Metode Adaptif

```r
riemann_adaptint(f, a, b, m=100)
```

```
## [1] 289.4
```

Metode Romberg

```r
romberg(f, a, b, m=10)
```

```
## [1] 289.4
```

Metode Monte Carlo

```r
monte_int(f, a, b)
```

```
## [1] 289.6
```

9.13 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press.
2. Chapra, S.C. Canale, R.P. 2015. **Numerical Methods For Engineers, Seventh Edition**. Mc Graw Hill.
3. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
4. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
5. Sanjaya, M. 2015. **Metode Numerik Berbasis Phython**. Penerbit Gava Media: Yogyakarta.
6. Suparno, S. 2008. **Komputasi untuk Sains dan Teknik Edisi II**. Departemen Fisika-FMIPA Universitas Indonesia.

9.14 Latihan

1. Hitung integral fungsi $f\left(x\right)=\sin^{2}\left(x\right)$ pada domain $x\in\left[0,\pi\right]$!
2. Tuliskan fungsi `R` yang dapat melakukan integrasi Riemann dengan aturan titik kiri !

3. Buatlah sebuah fungsi `R` yang dapat melakukan integrasi adaptif menggunakan metode Simpson 1/3 !
4. Kerjakan kembali soal 3 dengan menggunakan metode Simpson 3/8! (**Note**: pembaca dapat melakukan pencarian algoritma di internet dan mentrasformasikannya menjadi sintaks `R`)
5. Fungsi `monte_int()` hanya mampu melakukan integrasi pada domain positif. Buatlah algoritma baru sehingga metode ini dapat melakukan integrasi pada domain negatif !

Chapter 10

Persamaan Diferensial

Persamaan diferensial merupakan persoalan matematis yang sering dijumpai dalam bidang teknik lingkungan. Sering kali suatu persamaan diferensial tidak dapat diselesaikan secara analitik sehingga diperlukan metode numerik untuk menyelesaikannya. Pada Chapter 10, kita akan membahas masalah-masalah dalam persamaan diferensial dan metode penyelesaiannya. Adapun yang akan dibahas pada Chapter 10 kali ini antara lain:

- *Initial value problems*
- Sistem persamaan diferensial
- Persamaan diferensial parsial

10.1 *Initial value problems*

Initial value problems merupakan permasalahan yang sering ditemukan pada proses dekomposisi zat kimia atau polutan dalam reaktor. Penyelesaiaan persamaan diferensial biasanya dipersulit dengan tidak tersedianya informasi yang cukup untuk menyelesaikannya.Sebuah persamaan diferensial $f'(x, \dots)$ merupakan hasil diferensiasi beberapa fungsi $f(x, \dots)$. Proses penyelesaian persamaan diferensial, dan menemukan nilai $f(x, \dots)$ untuk beberapa nilai x,... tidak dimungkinkan karena integral dari $f'(x, \dots)$ hanya digambarkan bentuk umum. Pergeseran vertikal, atas atau bawah, tidak diketahui. Pergeseran vertikal ini menghasilkan konstanta integrasi.

Selama proses diferensiasi, nilai apapun dari proses pergeseran vertikal (integrasi) akan hilang sebagai akibat dari eliminasi konstanta yang memiliki turunan 0. Kita biasa melakukannya ketika mengintegrasikan fungsi dengan menambahkan konstanta $+C$ pada proses integrasi ke integral yang tidak terbatas.

Hal ini terkadang bukan menjadi permasalahan sebab jika menemukan nilai integrasi pada suatu batas tertentu syarat $+C$ dibatalkan dan konstanta integrasi tidak diperlukan.

Untuk persamaan diferensial biasa, tidak ada pembatalan yang nyaman, yang mengarah ke *initial value problems*. *Initial value problems* memberikan nilai $f(x_0, \ldots)$, di mana x_0 biasanya bernilai 0, meski tidak diharuskan. Nilai awal ini memberikan informasi yang cukup untuk menyelesaikan persamaan dan menemukan nilai aktual dari $f(x, \ldots)$ untuk sejumlah nilai x. Terdapat beberapa metode yang akan dibahas pada Chapter 10.1, antara lain:

- Metode Euler
- Metode Heun
- Metode Titik Tengah
- Metode Runge-Kutta Orde 4
- Metode multistep linier

10.1.1 Metode Euler

Metode Euler merupakan metode paling sederhana yang diturunkan dari deret Taylor. Penyelesaian *initial value problems* menggunakan metode Euler dilakukan melalui Persamaan (10.1).

$$y_{i+1} = y_i + f(x_i, y_i) h \tag{10.1}$$

dimana i merupakan tahapan iterasi.

Algoritma Metode Euler

1. Tentukan titik awal integrasi x_0 dan y_0.
2. Tentukan jumlah iterasi n dan *step size* h yang digunakan.
3. Lakukan integrasi menggunakan Persamaan (10.1).

Algoritma tersebut, selanjutnya dapat disusun ke dalam sebuah fungsi `R`. Fungsi tersebut adalah sebagai berikut:

```
euler <- function(f, x0, y0, h, n){
  x <- x0
  y <- y0

  for(i in 1:n){
    y0 <- y0 + h*f(x0, y0)
    x0 <- x0 + h
    x <- c(x,x0)
    y <- c(y, y0)
  }

  return(data.frame(x=x, y=y))
}
```

Contoh 10.1. Selesaikan persamaan diferensial di bawah ini, jika diketahui f(0)=1 menggunakan h=0,05 dan n=100!

$$f'\left(x,y\right)=\frac{y}{2x+1}$$

Jawab:

Penyelesaian secara analitik persamaan tersebut untuk nilai $f\left(0\right)=1$ sebagai berikut:

$$f\left(x\right)=\sqrt{2x+1}$$

Secara numerik persamaan tersebut dapat diselesaikan sebagai berikut:

iterasi 1

$$y\left(0,5\right)=1+0,05\times\frac{1}{\left(2\cdot 0\right)+1}=1,05$$

iterasi 2

$$y\left(0,1\right)=1,05+0,05\times\frac{1}{\left(2\cdot 0,05\right)+1}=1.097727$$

Kita dapat juga menggunakan fungsi `euler()` untuk menyelesaikan persamaan tersebut secara numerik. Hasil yang diperoleh selanjutnya diplotkan dengan hasil yang diperoleh menggunakan metode analitik. Berikut adalah sintaks yang digunakan:

```
# metode numerik
f1 <- function(x,y){y/(2*x+1)}
num <- euler(f1, x0=0, y0=1, h=0.05, n=100)

# metode analitik
f2 <- function(x){sqrt(2*x+1)}
x0 <- 0
y0 <- 1
x <- x0
y <- y0

for(i in 1:100){
  y0 <- f2(x0+0.05)
  x0 <- x0+0.05
  x <- c(x, x0)
  y <- c(y, y0)
}
true <- data.frame(x=x, y=y)
```

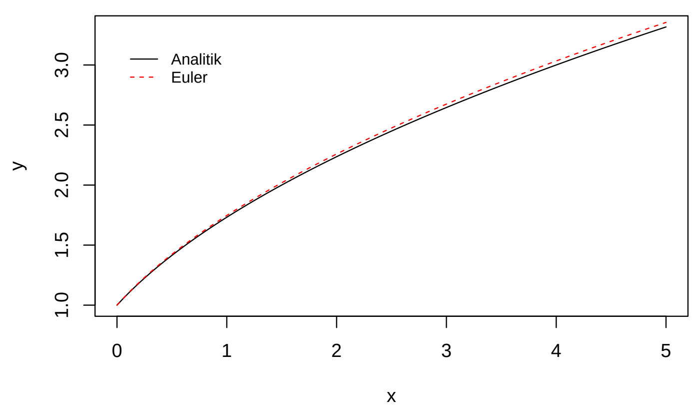

Figure 10.1: Visualisasi integrasi numerik dengan metode Euler dan metode analitik

Berdasarkan hasil visualisasi dapat dilihat bahwa metode Euler dapat dengan baik memberikan pendekatan nilai integrasi persamaan. Pembaca dapat mencoba untuk melakukan simulasi kembali dengan nilai h yang lebih kecil.

10.1.2 Metode Heun

Metode Heun merupakan salah satu peningkatan dari metode Euler. Metode ini melibatkan 2 buah persamaan. Persamaan pertama disebut sebagai persamaan prediktor yang digunakan untuk memprediksi nilai integrasi awal (Persamaan (10.2)). Persamaan kedua disebut sebagai persamaan korektor yang mengoreksi hasil integrasi awal (Persamaan (10.3)). Metode Heun pada *Chapter* ini merupakan metode prediktor-korektor satu tahapan. Akurasi integrasi dapat ditingkatkan dengan melakukan koreksi ulang terhadap nilai koreksi semula menggunakan persamaan kedua.

$$y_{i+1}^{0} = y_i + f\left(x_i, y_i\right) h \tag{10.2}$$

$$y_{i+1} = y_i + \frac{f\left(x_i, y_i\right) + f\left(x_{i+1}, y_{i+1}^{0}\right)}{2} h \tag{10.3}$$

Algoritma Metode Heun

1. Tentukan titik awal integrasi x_0 dan y_0.
2. Tentukan jumlah iterasi n dan *step size* h yang digunakan.
3. Lakukan prediksi nilai awal dengan Persamaan (10.2).
4. Lakukan koreksi nilai awal menggunakan Persamaan (10.3).
5. Lakukan koreksi terhadap nilai koreksi yang dihasilkan sebelumnya menggunakan Persamaan (10.3).

Kita dapat membangun sebuah fungsi yang dapat melakukan proses integrasi menggunakan metode Heun. Berikut adalah sintaks yang digunakan:

```
heun <- function(f, x0, y0, h, n, iter=1){
  x <- x0
  y <- y0

  for(i in 1:n){
    ypred0 <- f(x0,y0)
    ypred1 <- y0 + h*ypred0
    ypred2 <- f(x0+h,ypred1)
    ykor <- y0 + h*(ypred0+ypred2)/2
    if(iter!=1){
      for(i in 1:iter){
```

```
        ykor <- y0 + h*(ypred0+f(x0+h,ykor))/2
      }
    }
    y0 <- ykor
    x0 <- x0 + h
    x <- c(x, x0)
    y <- c(y, y0)
  }

  return(data.frame(x=x,y=y))
}
```

Contoh 10.2. Selesaikan kembali persamaan yang ditampilkan pada Contoh 10.1 menggunakan metode Heun!

Jawab:

Contoh perhitungan secara manual menggunakan metode Heun untuk sekali iterasi adalah sebagai berikut:

$$f'(0;1) = \frac{1}{(2 \cdot 0) + 1} = 1$$

$$y_1^0 = 1 + 0,05 \cdot 1 = 1,05$$

$$y_1' = f'(0,05;1,05) = \frac{1,05}{(2 \cdot 0,05) + 1} = 0,9545455$$

$$y_1 = 1 + 0,05 \cdot \frac{1 + 0,9545455}{2} = 1,047727$$

Penyelesaian persamaan tersebut menggunakan fungsi `heun()` dengan iterasi pada nilai koreksi sebanyak 1 kali disajikan pada sintaks berikut:

```
num <- heun(f1, x0=0, y0=1, h=0.05, n=100)
```

10.1.3 Metode Titik Tengah

Metode titik tengah menggunakan setengah *step size* pada metode Euler untuk melakukan estimasi terhadap integral suatu persamaan diferensial. Metode ini melakukan perhitungan melalui dua tahapan yaitu: menghitung nilai estimasi

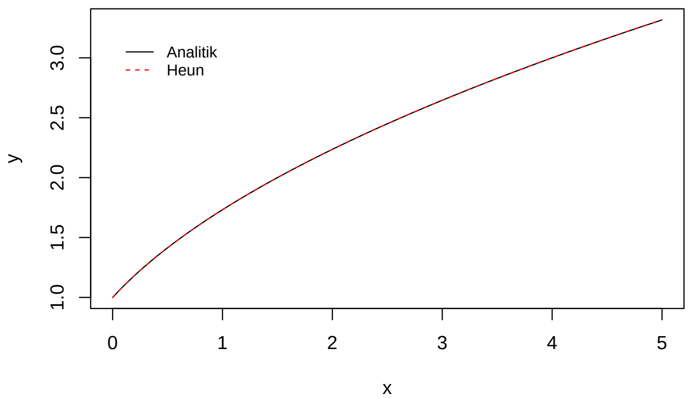

Figure 10.2: Visualisasi integrasi numerik dengan metode Heun dan metode analitik

integral pada setengah *step size*(Persamaan (10.4)) dan menghitung nilai integral menggunkan hasil perhitungan setengah *step size* sebelumnya (Persamaan (10.5)).

$$y_{i+\frac{1}{2}} = y_i + f\left(x_i, y_i\right)\frac{h}{2} \tag{10.4}$$

$$y_{i+1} = y_i + f\left(x_{i+\frac{1}{2}}, y_{i,\frac{1}{2}}\right)h \tag{10.5}$$

Algoritma Metode Tengah

1. Tentukan titik awal integrasi x_0 dan y_0.
2. Tentukan jumlah iterasi n dan *step size* h yang digunakan.
3. Lakukan integrasi pada setengah tahapan iterasi menggunakan Persamaan (10.4).
4. Lakukan iterasi pada setengah tahapan selanjutnya menggunakan Persamaan (10.5).

Berdasarkan algoritma tersebut, kita dapat membangun sebuah fungsi pada `R` yang adapat digunakan untuk menyelesaikan persamaan diferensial menggunakan metode titik tengah. Berikut adalah sintaks yang digunakan:

```
midpt <- function(f, x0, y0, h, n){
  x <- x0
  y <- y0

  for(i in 1:n){
    s1 <- y0 + f(x0,y0) * h/2
    s2 <- h * f(x0+h/2,s1)
    y0 <- y0 + s2
    x0 <- x0 + h
    x <- c(x, x0)
    y <- c(y, y0)
  }

  return(data.frame(x=x,y=y))
}
```

Contoh 10.3. Selesaikan kembali persamaan yang ditampilkan pada Contoh 10.1 menggunakan metode titik tengah!

Jawab:

Contoh perhitungan secara manual menggunakan metode titik tengah untuk sekali iterasi adalah sebagai berikut:

$$y_{\frac{1}{2}} = 1 + \frac{1}{(2 \cdot 0) + 1} \cdot \frac{0,05}{2} = 1,025$$

$$y_1 = 1 + \frac{1,025}{(2 \cdot 0,025) + 1} \cdot 0,05 = 1,0488$$

Penyelesaian persamaan tersebut menggunakan fungsi `midpt()` dengan iterasi pada nilai koreksi sebanyak 1 kali disajikan pada sintaks berikut:

```
num <- midpt(f1, x0=0, y0=1, h=0.05, n=100)
```

10.1.4 Metode Runge-Kutta Orde 4

Runge-Kutta orde 4 merupakan metode yang paling populer dalam penyelesaian persamaan diferensial. Metode ini dapat memperoleh akurasi deret Taylor tanpa memerlukan diferensiasi orde yang lebih tinggi. Metode Runge-Kutta orde 4 dituliskan ke dalam Persamaan (10.6).

$$y_{i+1} = y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) h \tag{10.6}$$

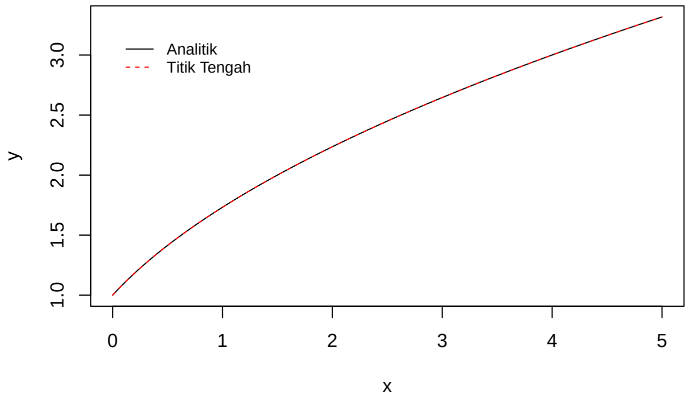

Figure 10.3: Visualisasi integrasi numerik dengan metode titik tengah dan metode analitik

dimana

$$k_1 = f\left(x_i, y_i\right) \tag{10.7}$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \tag{10.8}$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \tag{10.9}$$

$$k_4 = f\left(x_i + h, y_i + k_3 h\right) \tag{10.10}$$

Algoritma Metode Tengah

1. Tentukan titik awal integrasi x_0 dan y_0.
2. Tentukan jumlah iterasi n dan *step size* h yang digunakan.
3. Lakukan integrasi menggunakan Persamaan (10.6).

Berdasarkan algoritma tersebut, kita dapat membangun sebuah fungsi pada R yang adapat digunakan untuk menyelesaikan persamaan diferensial menggunakan metode Runge-Kutta orde 4. Berikut adalah sintaks yang digunakan:

```
rk4 <- function(f, x0, y0, h, n){
  x <- x0
  y <- y0

  for(i in 1:n){
    k1 <- f(x0,y0)
    k2 <- f(x0+0.5*h,y0+0.5*k1*h)
    k3 <- f(x0+0.5*h,y0+0.5*k2*h)
    k4 <- f(x0+h,y0+k3*h)
    y0 <- y0 + (1/6)*(k1+2*k2+2*k3+k4)*h
    x0 <- x0 + h
    x <- c(x, x0)
    y <- c(y, y0)
  }

  return(data.frame(x=x,y=y))
}
```

Contoh 10.4. Selesaikan kembali persamaan yang ditampilkan pada Contoh 10.1 menggunakan metode Runge-Kutta orde 4!

Jawab:

Contoh perhitungan secara manual menggunakan metode titik tengah untuk sekali iterasi adalah sebagai berikut:

$$k_1 = \frac{1}{(2 \cdot 0) + 1} = 1$$

$$k_2 = \frac{1 + \frac{1}{2} \cdot 1 \cdot 0,05}{\left(2 \cdot 0 \cdot \frac{1}{2} \cdot 0,05\right) + 1} = 1,025$$

$$k_2 = \frac{1 + \frac{1}{2} \cdot 1,025 \cdot 0,05}{\left(2 \cdot 0 \cdot \frac{1}{2} \cdot 0,05\right) + 1} = 1,025625$$

$$k_2 = \frac{1 + 1,025625 \cdot 0,05}{(2 \cdot 0 \cdot 0,05) + 1} = 1,051281$$

$$y_1 = 1 + \frac{1}{6}\left(1 + 2 \cdot 1,025 + 2 \cdot 1,025625 + 1,051281\right) 0,05 = 1.051271$$

```
1+(1/6)*(1+2*1.025+2*1.025625+1.051281)*0.05
```

```
## [1] 1.051
```

Iterasi dapat pula dilakukan dengan menggunakan fungsi `rk4()`. Berikut adalah sintaks yang digunakan:

```
num <- rk4(f1, x0=0, y0=1, h=0.05, n=100)
```

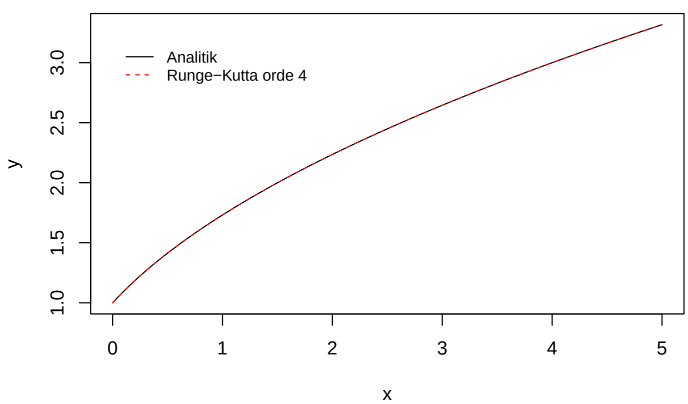

Figure 10.4: Visualisasi integrasi numerik dengan metode Runge-Kutta orde 4 dan metode analitik

10.1.5 Metode Multistep Linier

Jika metode Runge-Kutta mengalami kesulitan karena terlalu banyak evaluasi fungsi yang digunakan, masuk akal untuk bertanya apakah kita dapat menggunakan kembali beberapa evaluasi fungsi sebelumnya, yang sudah kita buat. Sebagai contoh, kita ingin tahu apakah kita dapat menggunakan kembali estimasi $f(0,1)$ dan $f(0,2)$ untuk memperkirakan nilai $f(0,3)$. Jika kita dapat menggunakan kembali perkiraan sebelumnya, kita dapat memperoleh akurasi tambahan tanpa menimbulkan penalti kinerja yang terkait dengan evaluasi fungsi tambahan. Metode multistep linier dikembangkan untuk mengatasi masalah ini.

Di satu sisi, metode multistep linier dasar untuk persamaan diferensial hanya mencakup satu titik x_i, dalam perhitungan $x_i + 1$. Ini persis bagaimana fungsi metode Euler dan metode Euler merupakan metode multistep linier dasar. Metode selanjutnya menggunakan $x_i - 1$ dan x_i untuk menghitung $x_i + 1$. Metode Adams-Bashforth menggunakan tambahan berbobot, termasuk bobot negatif, dari langkah dan poin untuk sampai pada langkah berikutnya. Seperti metode numerik lainnya, bobot muncul dari interpolasi polinomial titik yang tersedia.

Metode Adam-Bashforth orde 2 didasarkan pada Persamaan (10.11).

$$y_{i+2} = y_{i+1} + \frac{h}{2}\left(3f\left(x_{i+1}, y_{i+1}\right) - f\left(x_i, y_i\right)\right) \tag{10.11}$$

Pendekatan ini melakukan interpolasi antara titik sebelumnya untuk memperkirakan titik ketiga dalam grup. Titik ketiga ini menjadi titik tengah dari iterasi berikutnya saat seluruh proses berlanjut. Karena nilai sebelumnya disimpan dan digunakan kembali, evaluasi fungsi tambahan tidak diperlukan. Jika metode Runge-Kutta dapat dibandingkan dengan tip-toeing melalui bidang vektor, maka metode Adams-Bashforth dapat sama dibandingkan dengan menjalankan melalui bidang vektor. Namun, ini tidak berarti metode Adams-Bashforth lebih unggul.

Algoritma Metode Tengah

1. Tentukan titik awal integrasi x_0 dan y_0.
2. Tentukan jumlah iterasi n dan *step size* h yang digunakan.
3. Lakukan pendekatan pada iterasi ke-1 menggunakan metode Euler.
4. Lakukan integrasi ke-2 sampai n menggunakan Persamaan (10.11).

Berdasarkan algoritma tersebut, kita dapat membangun sebuah fungsi pada `R` yang adapat digunakan untuk menyelesaikan persamaan diferensial menggunakan metode multistep linier. Berikut adalah sintaks yang digunakan:

```
adambashforth <- function(f, x0, y0, h, n){
  # pendekatan Euler untuk x1 dan y1
  y1 <- y0 + h*f(x0,y0)
  x1 <- x0 + h

  x <- c(x0,x1)
  y <- c(y0,y1)
  n <- n-1
```

```
  for(i in 1:n){
    yn <- y1 + 1.5*h*f(x1,y1) - 0.5*h*f(x0,y0)
    xn <- x1 + h

    y0 <- y1
    x0 <- x1
    y1 <- yn
    x1 <- xn

    y <- c(y,y1)
    x <- c(x,x1)
  }

  return(data.frame(x=x,y=y))
}
```

Contoh 10.5. Selesaikan kembali persamaan yang ditampilkan pada Contoh 10.1 menggunakan metode Runge-Kutta orde 4!

Jawab:

Untuk melakukan iterasi, kita dapat menggunakan fungsi `adambashforth()`. Berikut adalah sintaks yang digunakan:

```
num <- rk4(f1, x0=0, y0=1, h=0.05, n=100)
```

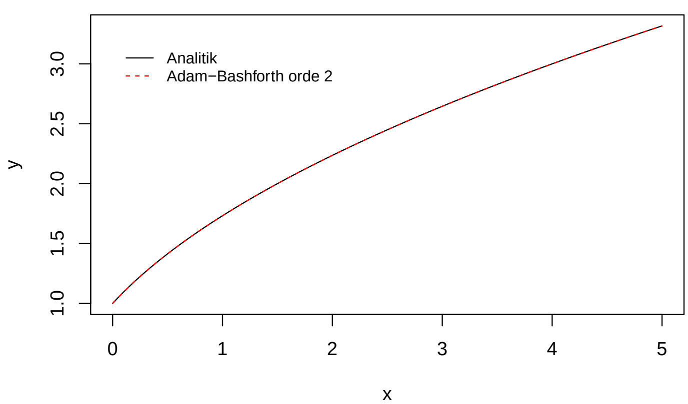

Figure 10.5: Visualisasi integrasi numerik dengan metode Adam-Bashfoth orde 2 dan metode analitik

10.2 Sistem Persamaan Diferensial

Sistem persamaan diferensial akan sering pembaca temui dalam pemodelan sistem dinamik. Pada proses pemodelan sistem tersebut akan ditemukan aksi-interasi antar komponennya yang dinyatakan ke dalam suatu sistem persamaan diferensial.

Pada fungsi `rk4sys()` disusun sebuah fungsi pada `R` untuk melakukan iterasi terhadap sistem persamaan diferensial menggunakan metode Runge-Kutta orde 4. Berikut adalah sintaks yang digunakan:

```
rk4sys <- function(f, x0, y0, h, n){
  x <- x0
  y <- y0

  values <- data.frame(x=x,t(y0))
  for(i in 1:n){
    k1 <- f(x0,y0)
    k2 <- f(x0+0.5*h,y0+0.5*k1*h)
    k3 <- f(x0+0.5*h,y0+0.5*k2*h)
    k4 <- f(x0+h,y0+k3*h)
    y0 <- y0 + (1/6)*(k1+2*k2+2*k3+k4)*h
    x0 <- x0 + h
    values <- rbind(values, data.frame(x=x0,t(y0)))
  }

  return(values)
}
```

Contoh 10.6. Sebuah model populasi yang dibagi menjadi tiga kelompok umur: anak (0-12 tahun), melahirkan anak (13-40 tahun), dan usia (41 tahun atau lebih). Kelompok 0-12 meningkat berdasarkan kelahiran di kelompok 13-40, dan menurun dengan kematian dan lewat bagian ke kelompok 13-40. Kelompok 13–40 meningkat dengan perolehan dari kelompok 0-12, dan berkurang dengan kematian dan lewat bagian ke dalam kelompok$>=41$. Kelompok$>=41$ meningkat dengan keuntungan dari kelompok 13-40, dan menurun dengan kematian. Parameter dipilih untuk mewakili tingkat kelahiran dan kematian yang cukup tinggi seperti yang ditemukan di banyak masyarakat berkembang. Model interaksi tersebut dinyatakan ke dalam persamaan interkasi di bawah ini! Simulasikan dinamika populasi pada model tersebut pada 10 tahun ke depan, jika diketahui populasi semula masing-masing kelompok usia secara berurutan adalah 200,400, dan 400, serta nilai masing-masing koefisien kelahiran, kematian kelompok 1, kematian kelompok2, dan kematian kelompok 3 secara berurutan adalah 0,5;0,1;0,1;0,25!

$$pop1' = b \cdot pop2' + \frac{11}{12} \cdot pop1' \left(1 - d1\right)$$

$$pop2' = \frac{1}{12} \cdot pop1' \cdot (1 - d1) + \frac{26}{27} \cdot pop2' \cdot (1 - d2)$$

$$pop3, = \frac{1}{27} \cdot pop2, \cdot (1 - d2) + pop3' \cdot (1 - d3)$$

Jawab:

Sistem persamaan diferensial perlu ditransformasi ke dalam bentuk sebuah fungsi pada `R`.

```
Population = function(x,y) {
   # parameter
   b = 0.5 # Birth rate in 13-40 group
   d1 = 0.1 # Death rate of 0-12 group
   d2 = 0.1 # Death rate of 13-40 group
   d3 = 0.25 # Death rate of 41 and older

   y1 = y[1] # 0-12 group population
   y2 = y[2] # 13-40 group population
   y3 = y[3] # 41 and older population
   y1.new = b*y2 + (11/12)*y1*(1 - d1)
   y2.new = (1/12)*y1*(1 - d1) + (26/27)*y2*(1 - d2)
   y3.new = (1/27)*y2*(1 - d2) + y3*(1 - d3)
   return(c(y1.new, y2.new, y3.new))
}
```

Nilai awal dituliskan seperti berikut:

```
# Nilai awal
y = c(pop1=200,pop2=400,pop3=400)
```

Simulasi model ditampilkan pada sintaks berikut:

```
pop <- rk4sys(Population, x0=0, y0=y, h=0.1, n=100)
head(pop)
```

```
##     x  pop1  pop2  pop3
## 1 0.0 200.0 400.0 400.0
## 2 0.1 239.0 437.9 432.6
## 3 0.2 283.4 479.6 467.9
```

```
## 4 0.3 334.0 525.4 506.1
## 5 0.4 391.4 575.9 547.4
## 6 0.5 456.4 631.3 592.1
```

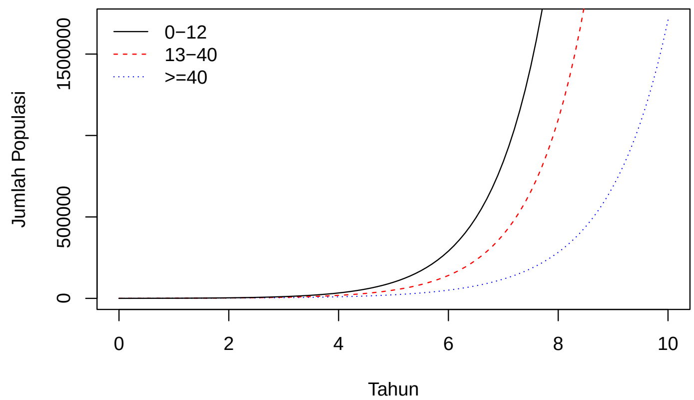

Figure 10.6: Visualisasi hasil simulasi model dinamika populasi

10.3 Penyelesaian Persamaan Diferensial dan Sistem Persamaan Diferensial Menggunakan Fungsi `ode()`

Fungsi `ode()` pada paket `deSolve` merupakan salah satu fungsi yang dapat digunakan untuk menyelesaikan persamaan dan sistem persamaan diferensial. Fungsi ini mudah digunakan serta menyediakan berbagai macam metode iterasi numerik untuk menyelesaikan persamaan diferensial. Format umum fungsi ini antara lain:

```
ode(y, times, func, parms,method , ...)
```

Catatan:

- `y` : nilai awal (kondisi awal) suatu persamaan diferensial.
- `times` : deret waktu yang terdiri dari kapan simulasi dimulai, kapan simulasi berkahir dan berapa *step size yang digunakan.*
- `func` : fungsi yang berisi persamaan diferensial. *Return value* pada fungsi haruslah berupa list.

- `parms` : list parameter yang diinputan kedalam `func`.
- `method` : sebuah string yang berupa metode intgrasi yang digunakan. Metode yang tersedia dan kegunaan metode tersebut antara lain:
  - "bdf": menangani persamaan diferensial menggunakan formula diferensiasi mundur (*bacward differensiation*) dan cocok untuk menangani kondisi *stiff*. Metode ini setara dengan `method="lsode"`.
  - "bdf_d": menggunakan formula diferensiasi mundur yang memanfaatkan iterasi Jacobi-Newton (mengabaikan elemen diagonal Jacobian). Cocok digunakan persamaan atau sistem persamaan diferensial kondisi *stiff*. Metode ini setara dengan `method="lsode",mf=23`.
  - "adams": metode Adams yang menggunakan iterasi fungsional (tanpa menggunakan Jacobian). Cocok digunakan untuk persamaan atau sistem persamaan *non stiff*. Metode ini setara dengan `method="lsode", mf=10`.
  - "impAdams": metode Adams implisit yang menggunakan iterasi Newton-Raphson. Metode ini setara dengan `method="lsode",mf=12`.
  - "impAdams_d": metode Adams implisit yang menggunakan iterasi Jacobi-Newton. Metode ini setara dengan `method="lsode, mf=13"`.
  - "euler": metode iterasi Euler
  - "rk4": metode iterasi Runge-Kutta orde 4.
  - metode lain: "lsoda", "lsode", "lsodes", "lsodar", "vode", "daspk", "ode23", "ode45", "radau".
- `...`: argumen tambahan untuk integrator atau method.

Contoh penerapan fungsi tersebut menggunakan Contoh 10.6, sebagai berikut:

```r
library(deSolve)

# sistem persamaan diferensial
Population = function(t,y,param) {
  y1 = y[1] # 0-12 group population
  y2 = y[2] # 13-40 group population
  y3 = y[3] # 41 and older population
  y1.new = b*y2 + 11/12*y1*(1 - d1)
  y2.new = 1/12*y1*(1 - d1) + 26/27*y2*(1 - d2)
  y3.new = 1/27*y2*(1 - d2) + y3*(1 - d3)
  return(list(c(y1.new, y2.new, y3.new)))
}
```

```
# parameter
b = 0.5 # Birth rate in 13-40 group
d1 = 0.1 # Death rate of 0-12 group
d2 = 0.1 # Death rate of 13-40 group
d3 = 0.25 # Death rate of 41 and older

# nilai awal
y = c(pop1=200, pop2=400, pop3=400)

# waktu
Tahun = seq(0,10,0.1)

# simulasi
out = ode(func = Population, y = y, times = Tahun,
          parms = c(b,d1,d2,d3), method = "rk4")
```

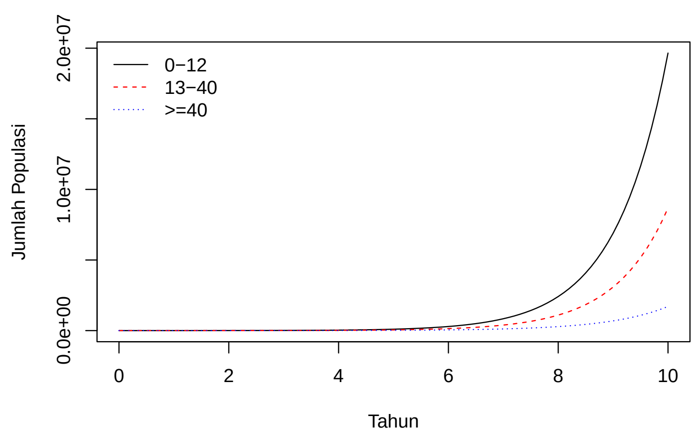

Figure 10.7: Visualisasi hasil simulasi model dinamika populasi menggunakan paket desolve

10.4 Persamaan Diferensial Parsial

Persamaan diferensial parsial (PDE) banyak dijumpai pada pemodelan transport polutan dalam bidang teknik lingkungan. Persamaan diferensial parsial merupakan persamaan diferensial yang melibatkan lebih dari satu variabel independen, biasanya variabel waktu dan satu atau lebih variabel posisi atau beberapa variabel spasial. PDE diklasifikasikan menjadi 3 jenis: parabolik (*time-*

dependent dan difusif), hiperbolik (*time-dependent* dan gelombang), dan eliptik (*time-independent*). Dalam penyelesaian PDE pada umumnya kita menggunakan metode FTCS (*forward in time, centered in space*). Untuk memahami definisi tersebut, pembaca dapat membaca kembali Chapter 9.1.

10.4.1 Persamaan Difusi

Persamaan difusi merupakan contoh PDE parabolik. Persamaan difusi satu dimensi spasial ditampilkan pada Persamaan (10.12).

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \tag{10.12}$$

Persamaan tersebut mirip dengan persamaan konduksi panas. Pada persamaan konduksi panas, variabel konsentrasi C diganti dengan variabel temperatur T, dan koefisien difusi D diganti dengan koefsien difusi termal K.

Untuk menyelesaikan Persamaan (10.12), persamaan tersebut diubah ke dalam bentuk metode beda hingga, dimana turunan waktu menggunakan pendekatan Euler (metode beda hingga maju) dan turunan variabel spasial diubah ke dalam bentuk pendekatan titik pusat. Proses diskretisasi Persamaan (10.12), ditampilkan pada Persamaan (10.13).

$$\frac{C\left(i+1,j\right)-C\left(i,j\right)}{\Delta t} = D\frac{C\left(i,j+1\right)+C\left(i,j-1\right)-2C\left(i,j\right)}{\Delta x^2} \tag{10.13}$$

dimana i merupakan *step* untuk variabel waktu t dan j merupakan *step* untuk variabel spasial x.

Persamaan (10.13) dapat disusun kembali sehingga menjadi Persamaan (10.14) yang menyatakan persamaan konsentrasi C pada saat $i+1$ pada posisi j.

$$C\left(i+1,j\right) = C\left(i,j\right) + A\left[C\left(i,j+1\right)+C\left(i,j-1\right)-2C\left(i,j\right)\right] \tag{10.14}$$

dimana

$$A = D\frac{\Delta t}{\Delta x^2} \tag{10.15}$$

Untuk stabilitas komputasi, pemilihan peningkatan waktu terhadap jarak yang dinyatakan pada nilai A harus $\leq \frac{1}{2}$.

Pada contoh berikut, kita akan melakukan simulasi menggunakan Persamaan (10.14). Parameter yang digunakan dan nilai awal yang digunakan dinyatakan pada sintaks berikut:

```
dt    <- 3                  # Timestep, s
dx    <- 0.1                # Distance step, cm
D     <- 1e-4               # Diffusion coeff, cm^2/s

# Cek apakah syarat stabilitas terpenuhi
D*dt/dx^2 <= 0.5
```

```
## [1] TRUE
```

```
# Desain grid points
L     <- 1                  # Length from -L/2 to L/2
n     <- L/dx + 1           # Number of grid points
x     <- seq(-L/2,L/2,dx) # Location of grid points
steps <- 30                 # Number of iterations
time  <- 0:steps
```

Langkah selanjutnya adalah inisiasi konsentrasi awal, dimana seluruh konsentrasi awal pada tiap grid adalah nol kecuali pada grid pusat.

```
C <- matrix(rep(0, (steps+1)*n), nrow = steps+1, ncol = n)
C[1, round(n/2)] <- 1/dx  # initial spike at central point
```

Simulasi selanjutnya dilakukan dengan melakukan iterasi pada *grid points* konsentrasi yang telah dibuat.

```
# Loop over desired number of time steps
for(i in 1:(steps-1)){
  for(j in 2:(n-1)){
    C[i+1,j] <- C[i,j] + (D*dt/dx^2)*(C[i,j+1] + C[i,j-1] - 2*C[i,j])
  }
}
```

Visualisasi dari hasil simulasi tersebut ditampilkan pada Gambar 10.8.

10.4.2 Persamaan Gelombang

Persamaan gelombang 1 dimensi ditampilkan pada Persamaan (10.16).

$$\frac{\partial^2 W}{\partial t^2} = c^2 \frac{\partial^2 W}{\partial x^2} \tag{10.16}$$

dimana W merupakan pemindahan dan c merupakan kecepatan gelombang. Persamaan (10.16) merupakan bentuk PDE hiperbolik. Versi lebih sederhana

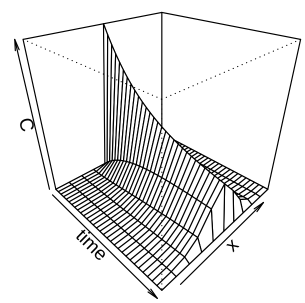

Figure 10.8: Visualisasi 3D simulasi difusi partikulat

dari Persamaan (10.16) adalah persamaan adveksi yang ditampilkan pada Persamaan (10.17).

$$\frac{\partial y}{\partial t} = -c\frac{\partial y}{\partial x} \tag{10.17}$$

Persamaan (10.17) menggambarkan evolusi pada bidang skalar $y\left(x, y\right)$ dibawa oleh gelombang dengan kecepatan konstan c dan bergerak dari kiri ke kanan jika $c > 0$. Persamaan adveksi merupakan contoh palin sederhana dari persamaan konservasi flux.

Diskretisasi Persamaan (10.17) ditampilkan pada Persamaan (10.18).

$$y\left(i+1, j\right) = y\left(i, j\right) - \frac{c\Delta t}{2\Delta x}\left[y\left(i, j+1\right) - y\left(i, j-1\right)\right] \tag{10.18}$$

Pada contoh berikut, kita akan melakukan simulasi menggunakan Persamaan (10.17). Parameter yang digunakan dan nilai awal yang digunakan dinyatakan pada sintaks berikut:

```
dt    <- 0.002               # Timestep, s
L     <- 1                   # Length from -L/2 to L/2
n     <- 50                  # Number of grid points
v     <- 1                   # Wavespeed, cm/s
dx    <- L/n                 # Distance step, cm
x     <- (1:n-0.5)*dx-L/2    # Location of grid points
steps <- L/(v*dt)            # Number of iterations
```

```
time  <- 0:steps
sig   <- 0.1                # Standard deviation of initial Gaussian wave
amp0  <- exp(-x^2/(2*sig^2))# Initial Gaussian amplitude
```

Kondisi awal dan kondisi batas periodik dinyatakan sebagai berikut:

```
C <- matrix(rep(0, (steps+1)*n), nrow = steps+1, ncol = n)
C[1, ] <- amp0
jplus1 <- c(2:n,1)
jminus1 <- c(n,1:(n-1))
```

Langkah selanjutnya melakukan interasi untuk melihat perubahan nilai pada *grid points*.

```
for(i in 1:steps){
  for(j in 1:n){
    C[i+1,j] <- C[i,j] + (v*dt/(2*dx))*(C[i,jplus1[j]] - C[i,jminus1[j]])
  }
}
```

Visualisasi dari hasil simulasi tersebut ditampilkan pada Gambar 10.9.

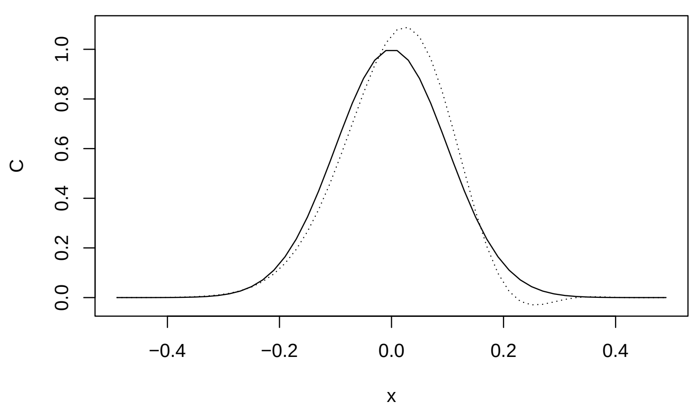

Figure 10.9: Visualisasi simulasi adveksi

10.4.3 Persamaan Laplace

Persamaan Laplace dalam dua dimensi disajikan pada Persamaan (10.19).

$$\frac{\partial V}{\partial x^2} + \frac{\partial V}{\partial y^2} = 0 \tag{10.19}$$

Persamaan (10.19) merupakan contoh tipe ketiga PDE, persamaan elips. Persoalan ini sering muncul dalam bidang elektrostatik, gravitasi, dan bidang lain di mana potensi V harus dihitung sebagai fungsi posisi. Jika ada muatan atau massa dalam ruang, dan jika kita menggeneralisasi ke tiga dimensi, persamaannya menjadi persamaan Poisson

$$\frac{\partial V}{\partial x^2} + \frac{\partial V}{\partial y^2} + \frac{\partial V}{\partial z^2} = 0 \tag{10.20}$$

Bergantung pada geometri masalahnya, Persamaan (10.20) juga dapat ditulis dalam koordinat bola, silindris, atau lainnya.

Untuk menyelesaikan persamaan eliptik jenis ini, persamaan harus diberi syarat batas. Biasanya dengan menentukan bahwa titik, garis, atau permukaan tertentu dipertahankan pada nilai konstan potensial. Kemudian potensi di titik lain disesuaikan sampai mencapai perkiraan yang diinginkan. (Dalam kasus yang jarang terjadi, persamaan dengan kondisi batas dapat diselesaikan dengan tepat secara analitis; tetapi biasanya solusi perkiraan harus cukup.)

Ada banyak pendekatan untuk solusi numerik dari persamaan Laplace. Mungkin yang paling sederhana adalah Jacobi, di mana titik-titik interior secara berturut-turut didekati dengan rata-rata titik-titik di sekitarnya, sedangkan titik-batas dibatasi pada nilai-nilai tetap dan yang ditentukan. Kita asumsikan sebagai contoh bidang persegi, dibatasi oleh $(0, 1)$ dalam arah x dan y, di mana ujung pada $y = 1$ dipertahankan pada $V = 1$ dan tiga tepi lainnya dipertahankan pada $V = 0$. Kita buat tebakan awal yang paling sederhana untuk potensi di titik interior, tetapi ini akan disamakan saat solusinya menyatu.

Pada contoh berikut, kita akan melakukan simulasi persamaan Laplace menggunakan metode Jacobi. Parameter yang digunakan dalam simulasi adalah sebagai berikut:

```
n   <- 30                 # Number of grid points/side
L   <- 1                  # Length of a side
dx  <- L/(n-1)            # Grid spacing
x   <- y    <- 0:(n-1)*dx # x and y coorfinates
```

Tebakan awal untuk profil voltase adalah sebagai berikut:

```
V0  <- 1
V   <- matrix(V0/2*sin(2*pi*x/L)*sin(2*pi*y/L),
              nrow = n, ncol = n, byrow = TRUE)
```

Kondisi batas di tetapkan seperti berikut:

```
V[1,] <- 0
V[n,] <- 0
V[,1] <- 0
V[,n] <- V0*rep(1,n)
```

Selanjutnya visualisasikan tebakan awal.

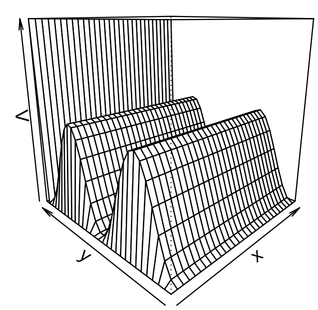

Figure 10.10: Visualisasi tebakan awal solusi persaman Laplace

Proses iterasi menggunakan metode Jacobi ditampilkan pada sintaks berikut:

```
newV  <- V
itmax <- n^2
tol   <- 1e-4
for(it in 1:itmax){
  dVsum <- 0
  for(i in 2:(n-1)){
    for(j in 2:(n-1)){
      newV[i,j] <- 0.25*(V[i-1,j]+V[i+1,j]+V[i,j-1]+V[i,j+1])
      dVsum <- dVsum + abs(1-V[i,j]/newV[i,j])
    }
  }
  V <- newV
  dV = dVsum/(n-2)^2 # Average deviation from prev. value
  if(dV < tol) break # Desired tolerance achived
}
```

```
it # Iterations to achieve convergence to tol
```

```
## [1] 419
```

```
dV
```

```
## [1] 9.908e-05
```

Hasil simulasi selanjutnya di visualisasikan kembali.

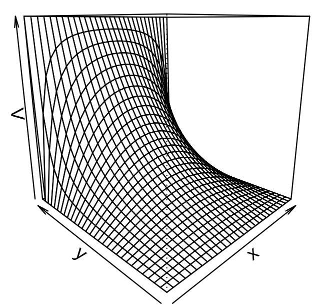

Figure 10.11: Visualisasi hasil simulasi solusi persaman Lplace

`## Penyelesaian Persamaan Diferensial Parsial Menggunakan Paket`ReacTran‘

Package `ReacTran` memfasilitasi pemodelan transport reaktif dalam 1, 2, dan 3 dimensi. Paket ini “berisi rutinitas yang memungkinkan pengembangan model transportasi reaktif dalam sistem perairan (sungai, danau, lautan), media berpori (agregat flok, sedimen, …) dan bahkan organisme yang diidealkan.

Pada paket `ReacTran` terdapat sejumlah fasilitas fungsi, antara lain:

- Fungsi untuk menyiapkan kisi beda hingga (1D atau 2D)
- Fungsi untuk melampirkan parameter dan properti ke kisi ini (1D atau 2D)
- Fungsi untuk menghitung jangka waktu transport adveksi-difusi di atas grid (1D, 2D, 3D)
- Berbagai fungsi lainnya.

Saat paket `ReacTran` dimuat, paket ini juga memuat dua paket pendukung, yaitu: `rootSolve` dan `deSolve`. Paket `rootSolve` berguna untuk memecahkan persamaan diferensial untuk kondisi tunak baik persamaan diferensial uniform atau multikomponen (1D, 2D, dan 3D). Sedangkan `deSolve` berguna untuk menyediakan fungsi yang digunakan untuk memperoleh penyelesaian numerik persamaan diferensial biasa (ODE), persamaan diferensial parsial (PDE), persamaan aljabar diferensial (DAE) dan persamaan *delay differential*.

10.4.4 `setup.grid.1D()`

Fungsi `setup.grid.1D()` digunakan untuk membentuk kisi satu dimensi. Secara sederhana fungsi ini membagi ruang satu dimensi L antara $x.up$ dan $x.down$ menjadi sejumlah N kisi sebesar `dx.1`. Format umum fungsi tersebut, adalah sebagai berikut:

```
setup.grid.1D(x.up=0, x.down=NULL,L=NULL,
              N=NULL, dx.1=NULL,
              p.dx.1= rep(1,length(L)),
              max.dx.1=L, dx.N=NULL,
              p.dx.N=rep(1,length(L)),
              max.dx.N=L)
```

Catatan:

- `x.up` : posisi hilir.
- `x.down` : posisi hulu.
- `L` : `x.down-x.up`.
- `N` : jumlah kisi = L/dx.1.

Pada situasi yang lebih kompleks, ukuran sel atau kisi dapat bervariasi, atau dapat pula lebih dari satu zona. Kondisi ini dijelaskan lebih jauh pada laman bantuan fungsi.

Nilai yang dihasilkan dari fungsi `setup.grid.1D()` termasuk `x.mid` (vektor sepanjang N yang menyatakan posisi titik tengah) dan `x.int` (vektor sepanjang $N + 1$ yang menyatakan posisi antar muka antara kisi sel dimana flux diukur).

Fungsi plot untuk `grid.1D()` memvisualissikan baik posisi sel dan ketebalan kotak, menampilkan `x.mid` dan `x.int`. Contoh di halaman bantuan menunjukkan perilaku ini.

`setup.grid.1D()` berfungsi sebagai titik awal untuk `setup.grid.2D`, yang membuat kisi di atas domain persegi panjang yang ditentukan oleh dua kisi 1D ortogonal.

10.4.5 setup.prop.1D()

Banyak model transportasi akan melibatkan kisi-kisi dengan properti konstan. Tetapi jika beberapa properti yang mempengaruhi difusi atau adveksi bervariasi dengan posisi di grid, variasi dapat digabungkan dengan fungsi `setup.prop.1D()` (atau `setup.prop.2D()` dalam dua dimensi).

Diberikan fungsi matematis atau matriks data, fungsi `setup.prop.1D()` menghitung nilai properti yang diminati di tengah sel kisi dan pada antarmuka antar sel. Format fungsi tersebut adalah sebagai berikut:

```
setup.prop.1D(func=NULL, value=NULL, xy=NULL,
              interpolate="spline",grid, ...)
```

Catatan:

- `func` : fungsi yang mengatur ketergantungan spasial properti.
- `value` : nilai konstan yang diberikan ke properti jika tidak ada ketergantungan spasial.
- `xy` : matriks data di mana kolom pertama memberikan posisi, dan kolom kedua memberikan nilai-nilai yang diinterpolasi di atas grid.
- `iterpolate` : metode interpolasi yang digunakan (spline atau linier).
- `grid`: objek yang dibentuk melalui fungsi `setup.grid.1D()`.
- `...`: argumen tambahan `func`.

10.4.6 tran.1D()

Fungsi ini menghitung istilah transportasi — laju perubahan konsentrasi akibat difusi dan adveksi — dalam model cairan 1D (fraksi volume = 1) atau padatan berpori (fraksi volume dapat bervariasi dan <1).`tran.1D()` juga digunakan untuk masalah dalam geometri bola atau silinder, meskipun dalam kasus ini antarmuka sel jaringan akan memiliki area variabel. Format fungsi ini adalah sebagai berikut:

```
tran.1D(C, C.up = C[1], C.down = C[length(C)],
        flux.up = NULL, flux.down = NULL,
        a.bl.up = NULL, a.bl.down = NULL, D = 0,
        v = 0, AFDW = 1, VF = 1, A = 1, dx,
        full.check = FALSE, full.output = FALSE)
```

Catatan:

- `C` : vektor konsentrasi pada titik tengah kisi sel.

- `C.up`, `C.down` : konsentrasi pada hulu dan hilir batas.
- `flux.up`,`flux.down` : flux dari dan keluar sistem pada hulu dan hilir batas.
- Jika terdapat tranport konvektif sepanjang hulu dan hilir batas, `a.bl.up` dan `a.bl.down` merupakan koefisiennya.
- `D`: Koefisien difusi.
- `v`: koefisien adveksi.
- `VF`: fraksi volume.
- `A`: fraksi area.
- `dx`: ketebalan kisi, baik nilai konstan atau vektor.
- `full.check`, `full.output`: nilai logik untuk memeriksa konsistensi dan mengatur output dari perhitungan. Keduanya `FALSE` secara default.

Ketika `full.output = FALSE`, nilai-nilai yang dikembalikan oleh `trans.1D()` adalah `dC`, yaitu: laju perubahan `C` di pusat setiap sel kisi karena transportasi, dan `flux.up` dan `flux.down`, yatu: fluks masuk dan keluar dari model di batas hulu dan hilir.

`ReacTran` juga memiliki fungsi untuk memperkirakan istilah difusi dan adveksi dalam model dua dan tiga dimensi, dan dalam koordinat silinder dan kutub. Jumlah input tumbuh dengan dimensi, tetapi input pada dasarnya sama seperti pada kasus 1D. Lihat halaman bantuan untuk `tran.2D()`, `tran.3D()`, `tran.cylindrical()`, dan `tran.polar()`.

Namun penyempurnaan lain adalah fungsi `tran.volume.1D()`, yang memperkirakan istilah transportasi volumetrik dalam model 1D. Berbeda dengan `tran.1D()`, yang menggunakan fluks (massa per satuan luas per satuan waktu), `tran.volume.1D()` menggunakan aliran (massa per satuan waktu). Ini berguna untuk memodelkan saluran yang area lintas bagiannya berubah, ketika perubahan area tidak perlu dimodelkan secara eksplisit. Ini juga memungkinkan input lateral dari saluran samping.

10.4.7 `ode.1D()` atau `steady.1D()`

Setelah kisi telah diatur dan properti ditetapkan dan model transport telah diformulasikan dengan `tran.1D()` (atau analog 2D atau 3D-nya), maka `ReacTran` memanggil `ode.1D()` dari paket `deSolve` jika solusi *time-dependent* diperlukan, atau `stable.1D()` dari paket `rootSolve` jika solusi kondisi-stabil diinginkan. Sistem ODE yang dihasilkan dari metode pendekatan garis biasanya *sparse* dan *non-stiff*. Integrator dalam `deSolve`, seperti “lsoda” (metode default 1D) sangat cocok untuk menangani sistem persamaan tersebut. Jika sistem ODE *non-stiff*, maka “adams” umumnya merupakan metode yang baik.

10.5 Contoh Penerapan Paket ReacTran

10.5.1 Persamaan Adveksi-Difuksi 1D

Pada contoh kali ini, kita akan memodifikasi persamaan difusi satu dimensi yang telah dilakukan sebelumnya dengan menambahkan bagian adveksi serta diselesaikan menggunakan paket `Reactran`.

Siapkan kisi menggunakan fungsi `setup.grid.1D()` dan persiapkan nilai parameter persamaan.

```
library(ReacTran)
N     <- 100          # Number of grid cells
xgrid <- setup.grid.1D(x.up = 0, x.down = 1, N = N)
x     <- xgrid$x.mid  # Midpoints of grid cells
D     <- 1e-4         # Diffusion coefficient
v     <- 0.1          # Advection velocity
```

Bentuk fungsi yang menyatakan persamaan adveksi-difusi.

```
Diffusion <- function(t, Y, parms) {
    tran<-tran.1D(C=Y,C.up=0,C.down=0,D=D,v=v,dx= xgrid)
    list(dY = tran$dC, flux.up = tran$flux.up,
        flux.down = tran $flux.down)
}
```

Inisiasi konsentrasi awal pada kisi sel.

```
Yini <- rep(0,N) # Initial concentration = 0
Yini[2] <- 100   # Except in the second cell
```

Lakukan perhitungandengn menggunakan *time step* 0,01.

```
# Calculate for 5 time units
times <- seq(from = 0, to = 5, by = 0.01)
out <- ode.1D(y = Yini, times = times, func = Diffusion,
             parms = NULL,dimens = N)
```

Visualisasikan hasil perhitungan.

10.5.2 Persamaan Gelombang 1D

Persamaan (10.16) dapat diselesaikan dengan cara yang sama dengan persamaan difusi dengan mengatur nilai $c^2 = D$, memisalkan $W = u$ dan $\frac{\partial u}{\partial t} = v$,

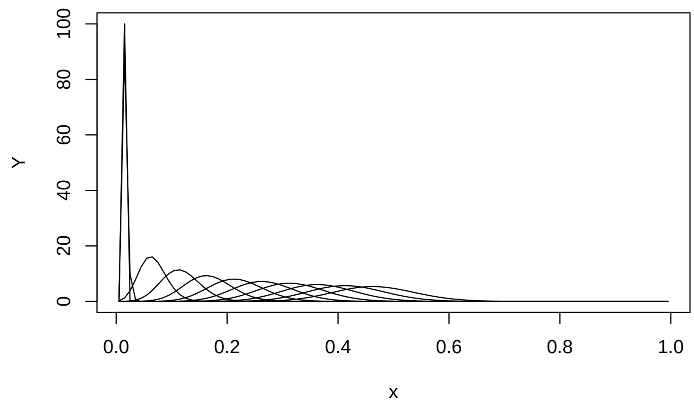

Figure 10.12: Visualisasi hasil simulasi persamaan adveksi-difusi menggunakan paket ReacTran

dan menyelesaikannya dengan cara yang familiar untuk variabel berpasangan (u, v). Di sini kita misalkan persamaan gelombang 1D untuk senar yang dipetik, yang awalnya ditahan pada 0 amplitudo untuk $x < -25$ dan $x > 25$, dan direntangkan secara linear hingga maksimum pada $x = 0$. `ode.1D()` digunakan untuk menyelesaikan set himpunan ODE simultan dengan $c = 1$.

Langkah pertama yang harus dilakukan adalah melakukan penetapan parameter.

```
dx    <- 0.2          # Spacing of grid cells
# String extends from -100 to +100
xgrid <- setup.grid.1D(x.up = -100,
                       x.down = 100, dx.1 = dx)
x     <- xgrid$x.mid  # midpoints of grid cells
N     <- xgrid$N      # number of grid cells
```

Menetapkan kondisi awal seperti ketinggian senar dan kecepatan.

```
uini  <- rep(0,N) # String height vector before stretching
vini  <- rep(0,N) # Initial string velocity vector
displ <- 10       # Initial displacement at center of string
# Impose initial triangular height profile on string between - 25
for(i in 1:N) {
  if (x[i] > -25 & x[i] <= 0) uini[i] = displ/25*(25 + x[i])else
  if (x[i] > 0 & x[i] < 25) uini[i] = displ/25*(25 - x[i])
```

```
}
yini  <- c(uini, vini)
```

Menetapkan deret waktu yang digunakan dalam simulasi.

```
times <- seq(from = 0, to = 50, by = 1)
```

Membangun fungsi yang akan diselesaikan secara numerik.

```
 wave <- function(t,y,parms) {
    u <- y[1:N] # Separate displacement and velocity vectors
    v <- y[(N+1):(2*N)]
    du<-v
    dv<-tran.1D(C=u,C.up=0,C.down=0,D=1,dx=xgrid)$dC
    return(list(c(du, dv)))
}
```

Selesaikan persamaan menggunakan fungsi `ode.1D()` dengan metode `adams`.

```
out <- ode.1D(func = wave, y = yini, times = times,
             parms = NULL, method = "adams",
             dimens = N, names = c("u", "v"))
u <- subset(out, which = "u") # Extract displacement vector
```

Visualisasikan hasil perhitungan.

10.5.3 Persamaan Laplace

Kita akan kembali melakukan simulasi terhadap Persamaan (10.19) menggunakan metode lainnya. Pada contoh kali ini, gradien pada sumbu y bernilai -1. Gradien hanyalah fluks, $D\,(\partial C = \partial x)$, dengan D set sama dengan 1. Fungsi penyelesaian yang digunakan adalah `steady.2D()`, karena tidak ada ketergantungan waktu dalam persamaan. Sebagai kondisi awal yang arbitrer, kami menggunakan nomor acak $N_x \times N_y$ yang terdistribusi secara seragam. Kita juga harus menentukan *nspec*, jumlah spesies dalam model (hanya satu, potensi, dalam hal ini), *dimens*, vektor dengan 2 nilai dengan jumlah sel dalam arah x dan y, dan `lrw`, panjang array. Lihat halaman bantuan untuk `steady.2D()` untuk informasi lebih detail.

Langkah pertama dalam melakukan simulasi adalah menetapkan parameter model.

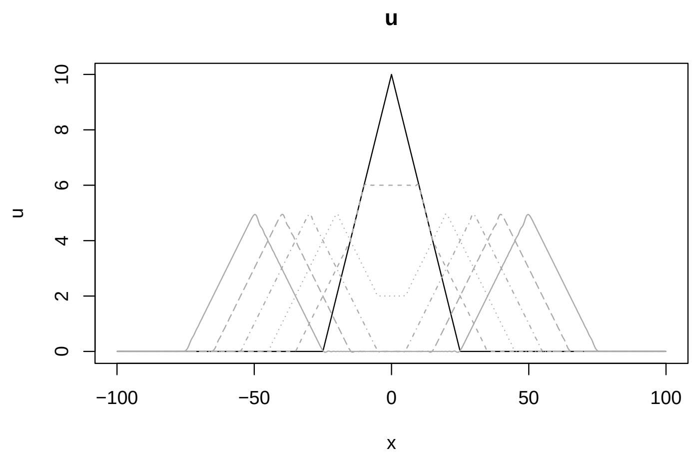

Figure 10.13: Visualisasi hasil simulasi persamaan gelombang menggunakan paket ReacTran

```
Nx    <- 100
Ny    <- 100
xgrid <- setup.grid.1D(x.up = 0, x.down = 1, N = Nx)
ygrid <- setup.grid.1D(x.up = 0, x.down = 1, N = Ny)
x     <- xgrid$x.mid
y     <- ygrid$x.mid
```

Bentuk fungsi yang akan diselesaikan secara numerik.

```
laplace <- function(t, U, parms) {
    w = matrix(nrow = Nx, ncol = Ny, data = U)
    dw = tran.2D(C = w, C.x.up = 0, C.y.down = 0,
    flux.y.up = 0,
    flux.y.down = -1,
    D.x = 1, D.y = 1,
    dx = xgrid, dy = ygrid)$dC
    list(dw)
}
```

Mulia dengan bilangan acak uniform sebagai kondisi awal, kemudian selesaikan untuk memperoleh nilai pada kondisi tunak dan buat plot kontur.

```
out = steady.2D(y = runif(Nx*Ny), func = laplace,
                  parms = NULL,nspec = 1,
                  dimens = c(Nx, Ny), lrw = 1e7)
```

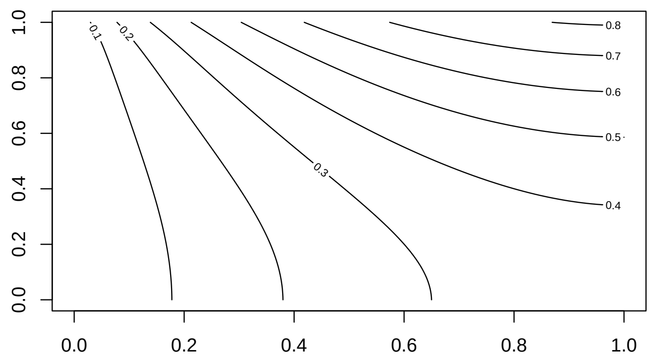

Figure 10.14: Visualisasi hasil simulasi persamaan laplace pada kondisi tunak menggunakan paket ReacTran

10.5.4 Persamaan Poisson untuk Dipol

Pada contoh kali ini, kita akan menyelesaikan persamaan Poisson untuk dipol yang ditampilkan pada Persamaan (10.21).

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} = -\frac{\rho}{\varepsilon_0} \tag{10.21}$$

untuk dipol yang terletak di tengah selembar persegi jika tidak pada 0 potensial. Untuk mempermudah, kita dapat mengatur semua faktor skala sama dengan satu. Dalam definisi fungsi poisson, nilai-nilai dalam matriks $N_x \times N_y$ w adalah input melalui vektor data U. Seperti dalam persamaan Laplace di atas, kita menetapkan nilai awal w pada sel-sel grid sama dengan angka acak uniform.

Langkah pertama dalam melakukan simulasi adalah menetapkan parameter model.

```
Nx    <- 100
Ny    <- 100
```

```
xgrid <- setup.grid.1D(x.up = 0, x.down = 1, N = Nx)
ygrid <- setup.grid.1D(x.up = 0, x.down = 1, N = Ny)
x     <- xgrid$x.mid
y     <- ygrid$x.mid
```

Cari nilai x dan y pada titik kisi yang mendekati $(0,4;0,5)$ untuk muatan positif dan $(0,6;0,5)$ untuk muatan negatif.

```
# x and y coordinates of positive and negative charges
ipos <- which.min(abs(x - 0.4))
jpos <- which.min(abs(y - 0.50))
ineg <- which.min(abs(x - 0.6))
jneg <- which.min(abs(y - 0.50))
```

Bentuk fungsi Poisson yang akan diselesaikan secara numerik.

```
poisson <- function(t, U, parms) {
    w = matrix(nrow = Nx, ncol = Ny, data = U)
    dw = tran.2D(C = w, C.x.up = 0, C.y.down = 0,
    flux.y.up = 0,
    flux.y.down = 0,
    D.x = 1, D.y = 1,
    dx = xgrid, dy = ygrid)$dC
    dw[ipos,jpos] = dw[ipos,jpos] + 1
    dw[ineg,jneg] = dw[ineg,jneg] - 1
    list(dw)
}
```

Selesaikan untuk kondisi tunak dan buat visualisasinya.

```
out <- steady.2D(y = runif(Nx*Ny),
                 func = poisson,
                 parms = NULL,
                 nspec = 1,
                 dimens = c(Nx, Ny),
                 lrw = 1e7)
```

10.6 Studi Kasus

10.6.1 Model Lotka-Voltera

Model Lotka-Voltera merupakan model yang menggambarkan interaksi antara predator dan mangsa. Pada studi kasus kali ini model yang akan digunakan

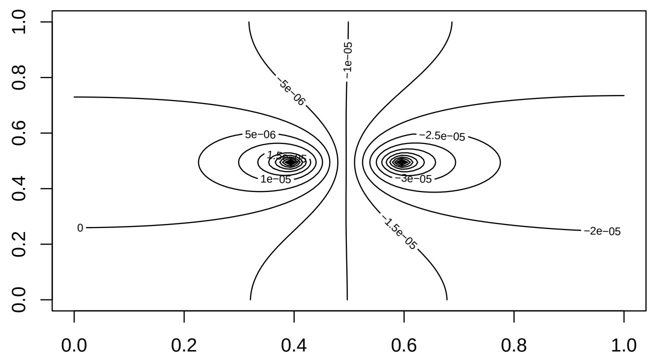

Figure 10.15: Visualisasi hasil simulasi persamaan Poisson pada kondisi tunak menggunakan paket ReacTran

merupakan model dengan 3 bentuk interaksi yaitu: tanaman u, herbivora v, dan karnivora w. Sistem persamaan diferensial sistem tersebut ditampilkan pada kumpulan persamaan berikut:

$$\frac{du}{dt} = au - \alpha_1 f_1 (u, v) \tag{10.22}$$

$$\frac{dv}{dt} = -bv + \alpha_1 f_1 (u, v) - \alpha_2 f_2 (v, w) \tag{10.23}$$

$$\frac{dw}{dt} = -c (w - w*) + \alpha_2 f_2 (v, w) \tag{10.24}$$

dimana $w*$ merupakan tingkat predasi minimum untuk menstabilkan populasi ketika populasi mangsa rendah. Interaksi antar komponen digambarkan ke dalam bentuk persamaan logistik.

$$f_i (x, y) = \frac{xy}{1 + k_i x} \tag{10.25}$$

Langkah pertama untuk menyelesaikan model adalah melakukan penentuan parameter model dan pembentukan fungsi model.

```
library(deSolve)
f <- function(x,y,k){x*y/(1+k*x)}
model <- function(t, xx, parms) {
    u = xx[1] # plant resource
    v = xx[2] # herbivore
    w = xx[3] # carnivore
    with(as.list(parms),{
    du = a*u - alpha1*f(u, v, k1)
    dv = -b*v + alpha1*f(u, v, k1) - alpha2*f(v, w, k2)
    dw = -c*(w - wstar) + alpha2*f(v, w, k2)
    list(c(du, dv, dw))
})}

times <- seq(0, 200, 0.1)
parms <- c(a=1, b=1, c=10, alpha1=0.2, alpha2=1,
k1<-0.05, k2=0, wstar=0.006)
xstart <- c(u=10, v=5, w=0.1)
```

Proses iterasi selanjutnya akan menggunakan metode `lsoda`, dimana metode iterasi ini dapat berganti secara otomatis dalam menangani sistem persamaan diferensial *stiff* dan *non-stiff*. Persamaan diferensial dikatakan *stiff* apabila variabel dependen berubah berdasarkan 2 atau lebih variabel independen yang sangat berbeda skalanya.

```
out = ode(xstart, times, model, parms,
          method ="lsoda")
```

Ketiga variabel selanjutnya divisualisasikan.

10.7 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press.
2. Chapra, S.C. Canale, R.P. 2015. **Numerical Methods For Engineers, Seventh Edition**. Mc Graw Hill.
3. Griffiths, G.W. 2016. **Numerical analysis using R : solutions to ODEs and PDEs**. Cambridge University Press.
4. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
5. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
6. Soetaert,K., Cash J., Mazzia F. 2012. **Solving Differential Equations in R**. Springer.

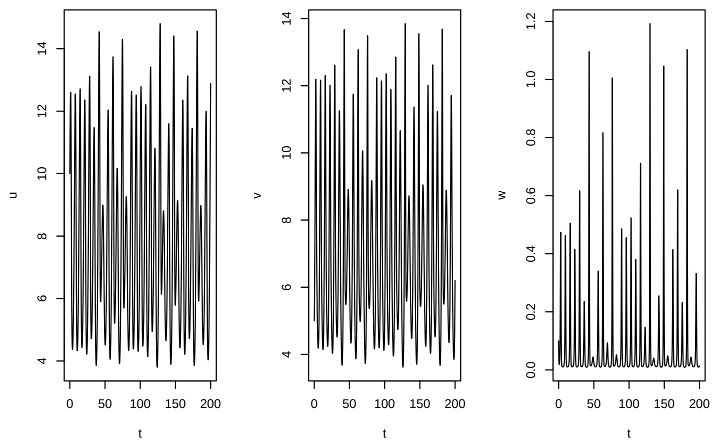

Figure 10.16: Visualisasi simulasi model Lotka-Voltera

7. Suparno, S. 2008. **Komputasi untuk Sains dan Teknik Edisi II**. Departemen Fisika-FMIPA Universitas Indonesia.

10.8 Latihan

1. Tunjukkan 10 hasil iterasi metode Euler untuk persamaan diferensial $f'(x, y) = y$ dimana $x_0 = 0$, $y_0 = 2$ dan *step size* $h = 0,1$!
2. Kerjakan lagi soal no.1 dengan menggunakan metode lainnya dan bandingkan seluruh metode tersebut menggunakan penyelesaian analitiknya!
3. Bentuk kembali fungsi `rk4sys()` menggunakan algoritma metode Adams-Bashforth dan namai fungsi tersebut `adamsys()`!

Chapter 11

Analisis Data

Pada Chapter 11, kita akan membahas mengenai cara melakukan analisis data pada `R`. Pada *Chapter* ini penulis akan memperkenalkan fungsi-fungsi yang ada pada `R` yang dapat membantu kita menganalisis data dan melakukan sejumlah uji statistik.

Pada *Chapter* ini kita tidak akan berfokus pada persamaan-persamaan matematika yang menjadi dasar suatu uji statistik. *Chapter* ini menitik beratkan pada bagaimana pembaca dapat melakukan sejumlah uji statistik pada `R` dan gambaran metode yang digunakan.

11.1 Import Data

Kita dapat melakukan import data dalam berbagai format pada `R`. Namun, pada *sub-chapter* ini hanya akan dibahas bagaimana cara mengimport data dari file dengan format `.csv`dan `.txt`. Secara umum fungsi-fungsi yang digunakan untuk membaca data pada file dengan format tersebut adalah sebagai berikut:

```
read.table(file, header = FALSE, sep = "", dec = ".",
           stringsAsFactors = default.stringsAsFactors())

read.csv(file, header = TRUE, sep = ",", dec = ".")

read.csv2(file, header = TRUE, sep = ";", dec = ",")

read.delim(file, header = TRUE, sep = "\t", dec = ".")

read.delim2(file, header = TRUE, sep = "\t", dec = ",")
```

Catatan:

- `file` : lokasi dan nama file yang akan dibaca diakhiri dengan format file. Secara *default* fungsi akan membaca file yang ada pada *working directory*. Untuk mengetahui lokasi *working directory*, jalankan fungsi `getwd()`. Salin file yang akan dibaca pada lokasi *working directory*.
- `header` : nilai logik yang menunjukkan apakah baris pertama pada file yang dibaca akan dibaca sebagai nama kolom.
- `sep` : simbol yang menujukkan pemisah antar data. Pemisah antar data dapat berupa "",";",".", dll.
- `dec` : simbol yang menujukkan desimal. Pemisah desimal dapat berupa "." atau ",".
- `stringsAsFactors` : nilai logik yang menunjukkan apakah jenis data `string` akan dikonversi menjadi `factor`.

Kelima fungsi tersebut digunakan untuk membaca data tabular atau data yang disusun kedalam format tabel. Fungsi `read.table()` merupakan bentuk umum dari keempat fungsi lainnya. Fungsi tersebut dapat digunakan untuk membaca data dalam kedua format yang telah disebutkan sebelumnya. Fungsi lainnya lebih spesifi, dimana fungsi `read.csv()` dan `read.csv2()` digunakan untuk membaca data dengan ekstensi `.csv`, sedangkan `read.delim()` dan `read.delim2()` untuk membaca data dengan ekstensi `.txt`. Berikut adalah contoh bagaimana cara membaca data dengan nama `data.csv` yang ada pada *working directory* dengan pemisah antar data berupa `;` dan tanda koma berupa `,`:

```
data <- read.table(file="data.csv", sep=";", dec=",")
```

11.2 Membaca Data Dari *Library*

Untuk keperluan pendidikan atau pengujian sebuah fungsi biasanya dalam sebuah *library* disediakan dataset yang siap digunakan. `R` melalui *library* `datasets` menyediakan sejumlah data yang dapat digunakan untuk berlatih menggunakan `R`. Berikut adalah fungsi yang digunakan untuk mengecek dataset apa saja yang tersedia pada sebuah *library*:

```
data(package=.packages(all.available = TRUE))
```

Catatan:

- `package`: nama *library* yang hendak dicek dataset yang tersedia.

Berikut adalah contoh cara melakukan pengecekan pada dataset yang tersedia pada *library* `datasets`:

```
data(package="datasets")

# cek seluruh dataset dari seluruh library yg telah dimuat
data()
```

11.3 Ringkasan Data

Terdapat sejumlah fungsi yang akan pembaca sering gunakan untuk mengecek dataset yang akan pembaca analisa. Fungsi-fungsi tersebut antara lain:

- `head()`: mengecek n (*default* 6) observasi teratas.
- `tail()`: mengecek n (*default* 6) observasi terbawah.
- `str()`: mengecek struktur data atau jenis data pada masing-masing kolom. Jenis data yang ada pada `R` dapat berupa `num` (numerik), `int` (integer), `Factor`(factor), `date` (tanggal), dan `chr` (karakter atau string).
- `summary()`: ringkasan data.

Berikut adalah contoh penerapan fungsi-fungsi tersebut pada dataset `iris`:

```
# cek 10 observasi teratas
head(iris, 10)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1           5.1         3.5          1.4         0.2
## 2           4.9         3.0          1.4         0.2
## 3           4.7         3.2          1.3         0.2
## 4           4.6         3.1          1.5         0.2
## 5           5.0         3.6          1.4         0.2
## 6           5.4         3.9          1.7         0.4
## 7           4.6         3.4          1.4         0.3
## 8           5.0         3.4          1.5         0.2
## 9           4.4         2.9          1.4         0.2
## 10          4.9         3.1          1.5         0.1
##    Species
## 1   setosa
## 2   setosa
## 3   setosa
## 4   setosa
## 5   setosa
## 6   setosa
```

```
## 7   setosa
## 8   setosa
## 9   setosa
## 10  setosa
```

```
# cek 10 observasi terbawah
tail(iris, 10)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 141          6.7         3.1          5.6         2.4
## 142          6.9         3.1          5.1         2.3
## 143          5.8         2.7          5.1         1.9
## 144          6.8         3.2          5.9         2.3
## 145          6.7         3.3          5.7         2.5
## 146          6.7         3.0          5.2         2.3
## 147          6.3         2.5          5.0         1.9
## 148          6.5         3.0          5.2         2.0
## 149          6.2         3.4          5.4         2.3
## 150          5.9         3.0          5.1         1.8
##       Species
## 141 virginica
## 142 virginica
## 143 virginica
## 144 virginica
## 145 virginica
## 146 virginica
## 147 virginica
## 148 virginica
## 149 virginica
## 150 virginica
```

```
# cek struktur data
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 .
```

```
# ringkasan data
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length
##  Min.   :4.30   Min.   :2.00   Min.   :1.00
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60
##  Median :5.80   Median :3.00   Median :4.35
##  Mean   :5.84   Mean   :3.06   Mean   :3.76
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10
##  Max.   :7.90   Max.   :4.40   Max.   :6.90
##   Petal.Width        Species
##  Min.   :0.1   setosa    :50
##  1st Qu.:0.3   versicolor:50
##  Median :1.3   virginica :50
##  Mean   :1.2
##  3rd Qu.:1.8
##  Max.   :2.5
```

Fungsi-fungsi lainnya yang dapat digunakan untuk melakukan analisis statistika deskriptif adalah sebagai berikut:

- `mean()` : menghitung nilai rata-rata variabel numerik.
- `sd()` : menghitung simpangan baku variabel numerik.
- `var()` : menghitung varians variabel numerik.
- `median()` : menghitung median suatu variabel numerik.
- `range()` : memperoleh nilai minimum dan maksimum suatu variabel numerik.
- `IQR()` : memperoleh nilai jarak antar kuartil.
- `quantile()` : memperoleh kuantil variabel numerik.

Berikut adalah contoh penerapan fungsi-fungsi tersebut:

```r
attach(airquality)

# rata-rata konsentrasi ozon
mean(Ozone, na.rm = TRUE)
```

```
## [1] 42.13
```

```r
# median konsentrasi ozon
median(Ozone, na.rm = TRUE)
```

```
## [1] 31.5
```

```r
# simpangan baku konsentrasi ozon
sd(Ozone, na.rm = TRUE)
```

```
## [1] 32.99
```

```
# varians konsentrasi ozon
var(Ozone, na.rm = TRUE)
```

```
## [1] 1088
```

```
# range konsentrasi ozon
range(Ozone, na.rm = TRUE)
```

```
## [1]   1 168
```

```
# IQR konsentrasi ozon
IQR(Ozone, na.rm = TRUE)
```

```
## [1] 45.25
```

```
# kuartil 1, 2 dan 3 konsentrasi ozon
quantile(Ozone, probs = c(0.25, 0.5, 0.75), na.rm = TRUE)
```

```
##   25%   50%   75% 
## 18.00 31.50 63.25
```

```
detach(airquality)
```

11.4 Uji Normalitas Data Tunggal

Pada analisis statistik inferensial khususnya pada pengujian hipotesis, asumsi normalitas merupakan sesuatu yang harus terpenuhi jika prosedur uji yang digunakan merupakan prosedur uji parametrik. Terdapat dua buah cara untuk melakukan uji tersebut, antara lain:

1. Metode grafis (qq-plot, ECDF, plot densitas, histogram, dan boxplot).
2. Metode matematis (Shapiro-Wilk, Cramer-von Mises, Shapiro-Francia, Anderson-Darling, Liliefors, Pearson Chi-square, dll).

Pada *Chapter* ini, kita akan berfokus pada uji matematis karena cara pengujian dengan menggunakan metode grafis telah penulis jabarkan pada *Chapter* visualisasi data.

Metode uji normalitas yang sering digunakan pada `R` adalah metode Shapiro-Wilk. Metode ini merupakan metode uji yang memiliki power yang besar khusunya untuk ukuran sampel yang relatif kecil. Versi awal metode ini terbatas dengan jumlah sampel 3 sampai 50 sampel. Versi selanjutnya mengalami modifikasi sehingga dapat menangani sampel sampai dengan 5000 sampel bahkan lebih.

Untuk melakukan uji SHapiro-Wilk pada `R`, pembaca dapat menggunakan fungsi `shapiro.test()`. Format fungsi tersebut adalah sebagai beriku:

```
shapiro.test(x)
```

Catatan:

- `x` : vektor numerik.

Untuk lebih memahami impelementasi fungsi tersebut pada data, berikut adalah contoh penerapan fungsi tersebut untuk menguji normalitas distribusi konsentrasi ozon pada dataset `airquality`:

```
shapiro.test(airquality$Ozone)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  airquality$Ozone
## W = 0.88, p-value = 3e-08
```

Berdasarkan hasil uji diperoleh nilai p-value < 0,05, sehingga H_0 ditolak dan dapat disimpulkan bahwa distribusi konsentrasi ozon tidak mengikuti distribusi normal. Untuk lebih memahami prosedur pengujian normalitas distribusi suatu data pembaca dapat membaca lebih lanjut pada tautan Environmental Data Modelin.

11.5 Uji Rata-Rata Satu dan Dua Sampel

Uji rata-rata satu sampel merupakan uji statistik untuk menguji apakah rata-rata suatu sampel berasal dari suatu populasi yang telah diketahui nilai rata-ratanya. Sedangkan uji rata-rata untuk dua populasi dilakukan untuk menguji apakah kedua selisis rata-rata populasi tersebut bernilai nol yang menujukkan bahwa kedua populasi tersebut memiliki nilai rata-rata yang sama. Uji rata-rata dua populasi dapat dilakukan untuk sampel independen (contoh: uji rata-rata performa dua buah IPAL) dan berpasangan (contoh: uji rata-rata input dan output IPAL).

Untuk melakukan uji rata-rata pada `R` dapat digunakan fungsi `t.test()` untuk uji parametrik dan `wilcox.test()` untuk melakukan uji non-parametrik *sign rank test*. Format fungsi-fungsi tersebut adalah sebagai berikut:

```
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)

wilcox.test(x, y = NULL,
            alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, conf.level = 0.95, ...)
```

Catatan:

- `x,y` : vektor numerik. Jika argumen `x` dan `y` diisikan maka uji hipotesis dilakukan untuk dua buah populasi.
- `alternative`: digunakan untuk menentukan jenis uji hipotesis apakah satu sisi("less" dan "greater"), atau dua sisi ("two.sided").
- `mu` : nilai rata-rata populasi atau nilai rata-rata selisih antar populasi jika dilakukan uji hipotesis terhadap dua populasi. Secara default nilainya 0.
- `paired` : nilai logikal yang menentukan apakah uji dua populasi digunakan untuk sampel berpasangan (TRUE) atau tidak (FALSE).
- `var.equal` : nilai logikal yang menunjukkan apakah varians kedua populasi diasumsikan sama atau berbeda.
- `conf.level` : tingkat kepercayaan. Secara default tingkat kepercayaan yang digunakan adalah 95%.

Berikut adalah contoh penerapan fungsi tersebut untuk uji hipotesis satu dan dua populasi:

```
# Uji hipotesis konsentrasi ozon = 40 ppm
# parametrik
t.test(x=airquality$Ozone, alternative = "two.sided",
       mu = 40)
```

```
##
##  One Sample t-test
##
```

```
## data:  airquality$Ozone
## t = 0.7, df = 115, p-value = 0.5
## alternative hypothesis: true mean is not equal to 40
## 95 percent confidence interval:
##  36.06 48.20
## sample estimates:
## mean of x
##     42.13
```

```
# nonparametrik
wilcox.test(x=airquality$Ozone, alternative = "two.sided",
       mu = 40)
```

```
##
##  Wilcoxon signed rank test with continuity
##  correction
##
## data:  airquality$Ozone
## V = 3188, p-value = 0.7
## alternative hypothesis: true location is not equal to 40
```

```
# Uji hipotesis dua populasi
dni3 <- dimnames(iris3)
ii <- data.frame(matrix(aperm(iris3, c(1,3,2)), ncol = 4,
                        dimnames = list(NULL, sub(" L.",".Length",
                                        sub(" W.",".Width", dni3[[2]])))),
    Species = gl(3, 50, labels = sub("S", "s", sub("V", "v", dni3[[3]]))))
# parametrik
t.test(x=iris$Sepal.Length[iris$Species=="setosa"],
       y=ii$Sepal.Length[iris$Species=="versicolor"])
```

```
##
##  Welch Two Sample t-test
##
## data:  iris$Sepal.Length[iris$Species == "setosa"] and ii$Sepal.Length[iris$Species == "versic
## t = -11, df = 87, p-value <2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.1057 -0.7543
## sample estimates:
## mean of x mean of y
##     5.006     5.936
```

```
# nonparametrik
wilcox.test(x=iris$Sepal.Length[iris$Species=="setosa"],
       y=ii$Sepal.Length[iris$Species=="versicolor"])
```

```
##
##  Wilcoxon rank sum test with continuity
##  correction
##
## data:  iris$Sepal.Length[iris$Species == "setosa"] and ii$Sepal.Length[iris$Species
## W = 168, p-value = 8e-14
## alternative hypothesis: true location shift is not equal to 0
```

Fungsi `t.test()` akan menghasilkan output berupa nilai t uji, derajat kebebasan (df), nilai p-value, rentang estimasi nilai rata-rata berdasarkan tingkat kepercayaan yang digunakan, serta estimasi nilai rata-rata sampel. Fungsi `wilcox.test()` akan menghasilkan dua buah output yaitu nilai W dan p-value berdasarkan nilai W yang dihasilkan.

11.6 Korelasi Antar Variabel

Pada sebuah analisa, kita sering kali tertarik untuk menganalisa hubungan atau korelasi antara satu variabel terhadap variabel lainnya. Pengamatan adanya korelasi antar variabel dapat dilakukan melalui visualisasi menggunakan *scatterplot* dan perhitungan matematis menggunakan metode Pearson untuk metode parametrik dan metode rangking Spearman dan Kendall untuk metode nonparametrik. Pada *Chapter* ini kita akan berfokus untuk melakukan uji korelasi menggunakan `R` menggunakan metode matematis.

Pada `R` uji korelasi dapat dilakukan dengan menggunakan fungsi `cor.test()`. Format fungsi tersebut adalah sebagai berikut:

```
cor.test(x, y,
         alternative = c("two.sided", "less", "greater"),
         method = c("pearson", "kendall", "spearman"),
         conf.level = 0.95)
```

> **Catatan:**
>
> - `x,y` : vektor numerik.
> - `alternative`: digunakan untuk menentukan jenis uji hipotesis apakah satu sisi("less" dan "greater"), atau dua sisi ("two.sided").
> - `method` : metode perhitungan korelasi yang digunakan.

- `conf.level` : tingkat kepercayaan. Secara default tingkat kepercayaan yang digunakan adalah 95%.

Berikut adalah penerapan fungsi `cor.test()` berdasarkan metode-metode yang telah disediakan pada fungsi tersebut:

```
# Pearson
cor.test(x = airquality$Ozone, y = airquality$Solar.R,
         alternative = "two.sided",
         method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  airquality$Ozone and airquality$Solar.R
## t = 3.9, df = 109, p-value = 2e-04
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1732 0.5021
## sample estimates:
##    cor
## 0.3483
```

```
# Kendall
cor.test(x = airquality$Ozone, y = airquality$Solar.R,
         alternative = "two.sided",
         method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  airquality$Ozone and airquality$Solar.R
## z = 3.7, p-value = 2e-04
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##    tau
## 0.2403
```

```
# Spearman
cor.test(x = airquality$Ozone, y = airquality$Solar.R,
         alternative = "two.sided",
         method = "spearman")
```

```
## Warning in cor.test.default(x = airquality$Ozone, y =
```

```
## airquality$Solar.R, : Cannot compute exact p-value
## with ties

##
##  Spearman's rank correlation rho
##
## data:  airquality$Ozone and airquality$Solar.R
## S = 148561, p-value = 2e-04
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##    rho
## 0.3482
```

Berdasarkan output yang dihasilkan, metode Pearson menghasilkan output berupa nilai t uji, derajat kebebasan, nilai p-value, rentang estimasi nilai korelasi berdasarkan tingkat kepercayaan, dan estimasi nilai korelasi. Metode Kendall dan Spearman disisi lai menghasilkan output berupa nilai z uji dan S untuk masing-masing metode serta nilai p-value berdasarkan nilai statistika uji dan estimasi koefisien korelasi.

11.7 Analisis Varians

Pada *sub-Chapter* sebelumnya penulis telah menjelaskan uji rata-rata untuk satu sampel dan dua sampel. Pada kenyataannya dalam sebuah percobaan laboratorium, kita tidak hanya membandingkan dua buah grup sampel saja, namun beberapa grup dan sejumlah faktor. Untuk menganalisa apakah variasi perlakuan pada kelompok sampel akan memberikan hasil yang berbeda-beda pada rata-rata tiap grup atau tidak diperlukan analisis varians untuk menganilisa variasi perlakuan atau faktor pada masing-masing grup. Analisis varians dapat dilakukan baik untuk satu faktor maupun dua faktor atau lebih. Untuk melakukannya pada `R`, kita dapat menggunakan fungsi `aov()` untuk analisis varians dengan metode parametrik dan `kruskal.test()` untuk analisis varians dengan menggunakan metode nonparametrik. Berikut adalah format kedua fungsi tersebut:

```r
aov(formula, data = NULL)

kruskal.test(formula, data)
```

> **Catatan:**
>
> - `formula` : formula model yang digunakan.
> - `data`: dataset yang akan digunakan.

Berikut adalah contoh penerapan kedua fungsi tersebut untuk melihat apakah terdapat beda pada rata-rata konsentrasi bulanan ozon menggunakan dataset `airquality`:

```
summary(aov(Ozone~Month, airquality))
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Month         1   3387    3387    3.17  0.078 .
## Residuals   114 121756    1068
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 37 observations deleted due to missingness
```

```
kruskal.test(Ozone~Month, airquality)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Ozone by Month
## Kruskal-Wallis chi-squared = 29, df = 4, p-value
## = 7e-06
```

Berdasarkan hasil yang diperoleh diketahui bahwa rata-rata konsentrasi bulanan ozon tidak sama tiap bulannya atau minimal terdapat satu bulan dimana konsentrasi ozonnya berbeda secara signifikan dengan konsentrasi ozon pada bulan-bulan lainnya. Untuk lebih memahami terkait analisis varians pada `R` dan cara membaca output kedua fungsi tersebut, pembaca dapat membaca tulisan pada halaman situs sthda.

11.8 Analisis Komponen Utama

Analisis komponen utama menggunakan transformasi ortogonal (umumnya nilai singular atau dekomposisi nilai eigen) untuk mengubah seperangkat variabel pengamatan yang mungkin berkorelasi menjadi seperangkat variabel tidak berkorelasi (ortogonal) yang disebut komponen utama. Transformasi didefinisikan sedemikian rupa sehingga komponen utama pertama memiliki varians setinggi mungkin (menyumbang variabilitas pada data sebanyak mungkin), dan masing-masing komponen berikutnya pada gilirannya memiliki varians tertinggi yang mungkin di bawah kendala, dimana komponen tersebut menjadi ortogonal ke komponen sebelumnya.

Dalam R, analisis komponen utama umumnya dilakukan dengan fungsi `prcomp()`. Format fungsi tersebut adalah sebagai berikut:

```
prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE,
       tol = NULL)
```

Catatan:

- `x` : data frame atau matriks kompleks numerik.
- `retx` : nilai logik yang mengindikasikan apakah variabel hasil rotasi perlu ditampilkan.
- `center` : nilai logik yang mengidikasikan apakah variabel perlu dilakukan pergeseran sehingga nilai rata-ratanya berpusat pada nilai nol.
- `scale` : nilai logik yang mengidikasikan apakah variabel perlu dilakukan penskalaan sebelum dilakukan analisis.
- `tol` : nilai toleransi yang menunjukkan batas nilai bagi komponen yang akan dipertahankan. Komponen yang dihilangkan jika simpangan bakunya kurang dari atau sama dengan `tol x simpangan baku pc1`.

Untuk memahami penerapan fungsi tersebut, kita akan melakukan simulasi menggunakan dataset `iris`. Output yang dihasilkan di bawah ini menunjukkan bagaimana empat variabel numerik ditransformasikan menjadi empat komponen utama. Penskalaan data mungkin tidak diperlukan dalam kasus ini, karena keempat pengukuran memiliki unit yang sama dan besarnya sama. Namun, penskalaan umumnya merupakan praktik yang baik.

```
iris_use <- iris[,-5] # menghilangkan variabel non-numerik
iris_pca <- prcomp(iris_use, scale. = TRUE)
iris_pca
```

```
## Standard deviations (1, .., p=4):
## [1] 1.7084 0.9560 0.3831 0.1439
##
## Rotation (n x k) = (4 x 4):
##                  PC1      PC2     PC3     PC4
## Sepal.Length  0.5211 -0.37742  0.7196  0.2613
## Sepal.Width  -0.2693 -0.92330 -0.2444 -0.1235
## Petal.Length  0.5804 -0.02449 -0.1421 -0.8014
## Petal.Width   0.5649 -0.06694 -0.6343  0.5236
```

Fungsi `summary` memberikan proporsi varians total yang dikaitkan dengan masing-masing komponen utama, dan proporsi kumulatif ketika masing-masing komponen ditambahkan. Kita melihat bahwa dua komponen pertama berperan lebih dari 95% dari total varians.

```
summary(iris_pca)
```

```
## Importance of components:
##                          PC1   PC2    PC3     PC4
## Standard deviation     1.71 0.956 0.3831 0.14393
## Proportion of Variance 0.73 0.229 0.0367 0.00518
## Cumulative Proportion  0.73 0.958 0.9948 1.00000
```

Histogram (hasil plot dalam analisis `prcomp`) secara grafis merekapitulasi proporsi varian yang disumbangkan oleh masing-masing komponen utama, sementara biplot menunjukkan bagaimana variabel awal diproyeksikan pada dua komponen utama pertama (Gambar 11.1). Ini juga menunjukkan koordinat dari masing-masing sampel dalam ruang (PC1, PC2). Satu spesies iris (yang berubahmenjadi setosa dari analisis kluster di bawah ini) secara jelas dipisahkan dari dua spesies lain dalam ruang koordinat ini.

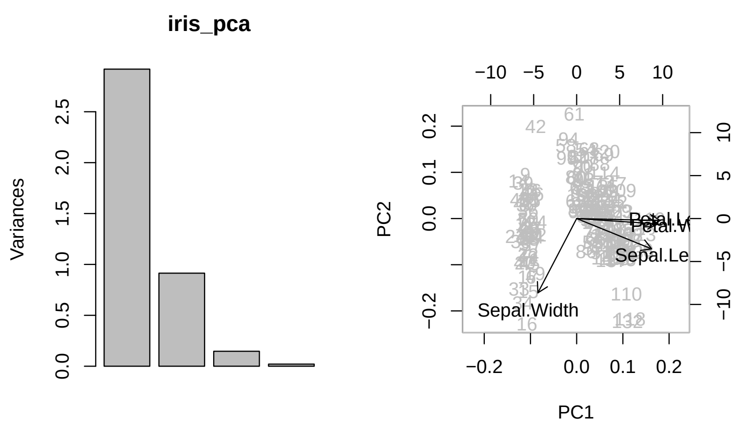

Figure 11.1: Analisis komponen utama data iris.

Untuk informasi lebih lanjut terkait metode analisis komponen utama, pembaca dapat membacanya pada laman Little Book of R for Multivariate Analysis.

11.9 Analisis Cluster

Analisis cluster mencoba untuk mengurutkan satu set objek ke dalam kelompok (cluster) sedemikian rupa sehingga objek dalam cluster yang sama lebih mirip satu sama lain dibandingkan objek pada cluster lainnya. Ini digunakan untuk analisis eksplorasi melalui proses penambangan data (*data mining*) di banyak bidang, seperti bioinformatika, biologi evolusi, analisis gambar, lingkungan, dan pembelajaran mesin.

Menurut Wikipedia: "Analisis Cluster itu sendiri bukanlah salah satu algoritma spesifik, tetapi tugas umum yang harus dipecahkan. Ini dapat dicapai dengan berbagai algoritma yang berbeda secara signifikan dalam pengertian mereka tentang apa yang merupakan sebuah cluster dan bagaimana cara menemukannya secara efisien. Gagasan populer mengenai cluster termasuk kelompok dengan jarak rendah di antara anggota cluster, area padat ruang data, interval atau distribusi statistik tertentu. Algoritma pengelompokan dan pengaturan parameter yang sesuai (termasuk nilai-nilai seperti fungsi jarak yang akan digunakan, ambang kepadatan atau jumlah cluster yang diharapkan) tergantung pada dataset individual dan tujuan penggunaan hasil. Analisis cluster seperti itu bukan tugas otomatis, tetapi proses berulang penemuan pengetahuan yang melibatkan *trial and error*. Seringkali diperlukan untuk memodifikasi preprocessing dan parameter sampai hasilnya mencapai properti yang diinginkan.

11.9.1 Analisis Cluster Menggunakan Algoritma Pengelompokan Hierarkis Aglomeratif

Hierarchical clustering membangun hierarki cluster, di mana metrik hierarki adalah suatu ukuran ketidaksamaan antar cluster. Menurut halaman bantuan untuk `hclust()`, metode pengelompokan hierarkis aglomeratif, "Fungsi ini melakukan analisis hierarki cluster menggunakan seperangkat ketidaksamaan untuk n objek yang dikelompokkan. Awalnya, masing-masing objek ditugaskan ke cluster sendiri dan kemudian algoritma melanjutkan secara iteratif, pada setiap tahap bergabung dengan dua cluster yang paling mirip, terus sampai hanya ada satu cluster. Pada setiap tahap, jarak antar kluster dihitung ulang dengan formula pembaruan ketidaksamaan Lance Williams sesuai dengan metode pengelompokan tertentu yang digunakan."Ada tujuh metode aglomerasi yang tersedia, dengan lengkap — yang mencari kluster kompak, bola — sebagai default. Format umum fungsi `hclust()` adalah sebagai berikut:

```
hclust(d, method = "complete", members = NULL)
```

> **Catatan:**
>
> - `d` : struktur ketidaksamaan yang dihasilkan dengan fungsi `dist`.
> - `method` : metode alglomerasi yang digunakan. Metode yang dapat digunakan antara lain: "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) atau "centroid" (= UPGMC)

- `members`: NULL atau vektor dengan ukuran sama dengan `d`.
  Untuk info lebih lanjut jalankan sintaks `?hclust`.

Untuk memahami penerapan fungsi `hclust()`, kita akan kembali menggunakan data `iris_use`. Berikut adalah sintaks yang digunakan:

```
# menghitung jarak antar observasi
iris_hclust <- dist(iris_use)

# Pembentukan Cluster
hc <- hclust(iris_hclust)
hc
```

```
##
## Call:
## hclust(d = iris_hclust)
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 150
```

Visualisasi cluster dibentuk melalui dendogram yang ditampilkan pada Gambar 11.2.

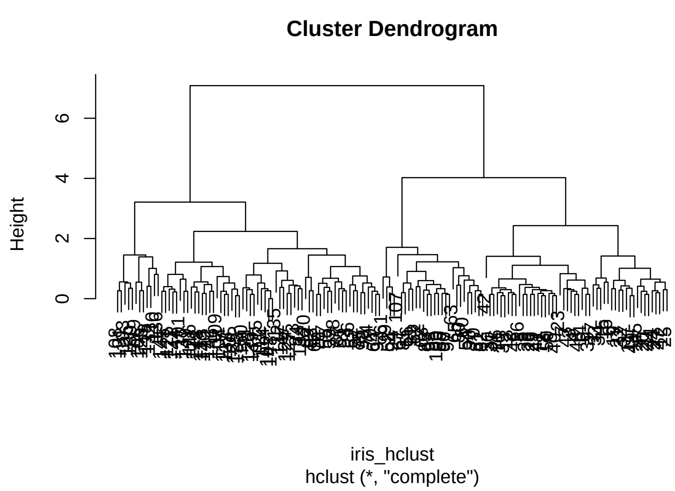

Figure 11.2: Analisis pengelompokan hierarkis alglomeratif data iris.

11.9.2 Pengelompokan Hierarkis Divisif

Menurut halaman bantuan `diana()` (*DIvisive ANAlysis Clustering*) dalam *library* `cluster`, “Algoritma diana membangun hierarki pengelompokan, dimulai dengan satu kluster besar yang berisi semua n pengamatan. Cluster dibagi sampai masing-masing cluster hanya berisi satu pengamatan. Pada setiap tahap, cluster dengan diameter terbesar dipilih. Format fungsi `diana()` adalah sebagai berikut:

```
diana(x, metric = "euclidean", stand = FALSE)
```

Catatan:

- `x` : struktur ketidaksamaan yang dihasilkan dengan fungsi `dist` atau data frame.
- `metric` : karakter string yang menyatakan metode pengukuran jarak yang digunakan. Metode pengukuran jarak dapat berupa “euclidean” dan “manhattan”.
- `stand`: vektor logik yang menyatakan apakah data akan dilakukan standardisasi terlebih dahulu sebelum dilakukan analisis.

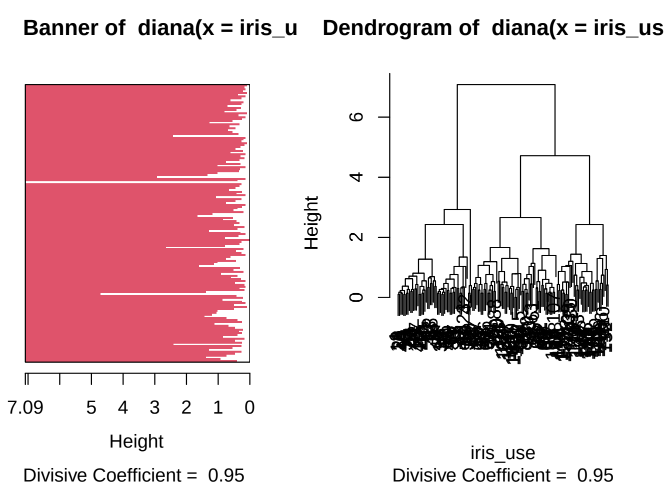

Figure 11.3: Analisis pengelompokan divisif data iris.

11.9.3 Pengelompokan Menggunakan Algoritma K-Mean

k-means melakukan pengelompokan n pengamatan ke dalam k cluster di mana setiap pengamatan akan tergabung dengan pusat cluster terdekat. Pengguna harus menentukan jumlah pusat (cluster) yang diinginkan sebagai output. Untuk melakukan pengelompokan dengan algoritma k-means pada `R` dapat menggunakan fungsi `kmeans()`. Format fungsi tersebut secara umum adalah sebagai berikut:

```
kmeans(x, centers, iter.max = 10,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                     "MacQueen"))
```

Catatan:

- `x` : data frame.
- `centers` : jumlah cluster yang ingin di buat.
- `iter.max`: jumlah iterasi maksimum yang diijinkan.
- `algorithm` : algoritma pengelompokan yang digunakan. Untuk informasi lebih lanjut jalankan sintaks `?kmeans`.

```
iris_kmeans <- kmeans(iris_use, centers = 3)
iris_kmeans
```

```
## K-means clustering with 3 clusters of sizes 50, 38, 62
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1        5.006       3.428        1.462       0.246
## 2        6.850       3.074        5.742       2.071
## 3        5.902       2.748        4.394       1.434
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [26] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [51] 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [76] 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [101] 2 3 2 2 2 2 3 2 2 2 2 2 2 3 3 2 2 2 2 3 2 3 2 3 2
## [126] 2 3 3 2 2 2 2 2 3 2 2 2 2 3 2 2 2 3 2 2 2 3 2 2 3
##
## Within cluster sum of squares by cluster:
## [1] 15.15 23.88 39.82
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"
## [4] "withinss"     "tot.withinss" "betweenss"
## [7] "size"         "iter"         "ifault"
```

```
# menghitung lokasi pusat cluster
ccent = function(cl) {
 f = function(i) colMeans(iris_use[cl==i,])
 x = sapply(sort(unique(cl)), f)
 colnames(x) = sort(unique(cl))
 return(x)
}

ccent(iris_kmeans$cluster)
```

```
##                  1     2     3
## Sepal.Length 5.006 6.850 5.902
## Sepal.Width  3.428 3.074 2.748
## Petal.Length 1.462 5.742 4.394
## Petal.Width  0.246 2.071 1.434
```

11.9.4 Pengelompokan Menggunakan Algoritma PAM

pam mem-partisi data menjadi k cluster di sekitar medoid. Medoid dari set data yang terbatas merupakan titik data dengan nilai ketidaksamaan rata-rata untuk semua titik data adalah minimum. Hal tersebut menujukkan bahwa medoid merupakan pusat dari set cluster. Menurut halaman bantuan `pam()`, pendekatan k-medoid lebih kuat daripada pendekatan k-means "karena meminimalkan jumlah ketidaksamaan daripada jumlah jarak euclidean kuadrat". Format umum fungsi `pam()` adalah sebagai berikut:

```
pam(x, k, diss = inherits(x, "dist"),
    metric = c("euclidean", "manhattan"))
```

> **Catatan:**
>
> - `x` : data frame.
> - `k` : jumlah cluster yang ingin di buat.
> - `method`: metode perhitungan jarak yang digunakan. Untuk informasi lebih lanjut jalankan sintaks `?pam`.

```
library(cluster)
iris_pam <- pam(iris_use, k=3)
iris_pam
```

```
## Medoids:
##       ID Sepal.Length Sepal.Width Petal.Length
## [1,]   8          5.0         3.4          1.5
```

```
## [2,]  79          6.0         2.9          4.5
## [3,] 113          6.8         3.0          5.5
##      Petal.Width
## [1,]         0.2
## [2,]         1.5
## [3,]         2.1
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [26] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [51] 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [76] 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [101] 3 2 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3
## [126] 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2
## Objective function:
##  build   swap
## 0.6709 0.6542
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering"
##  [4] "objective"  "isolation"  "clusinfo"
##  [7] "silinfo"    "diss"       "call"
## [10] "data"
```

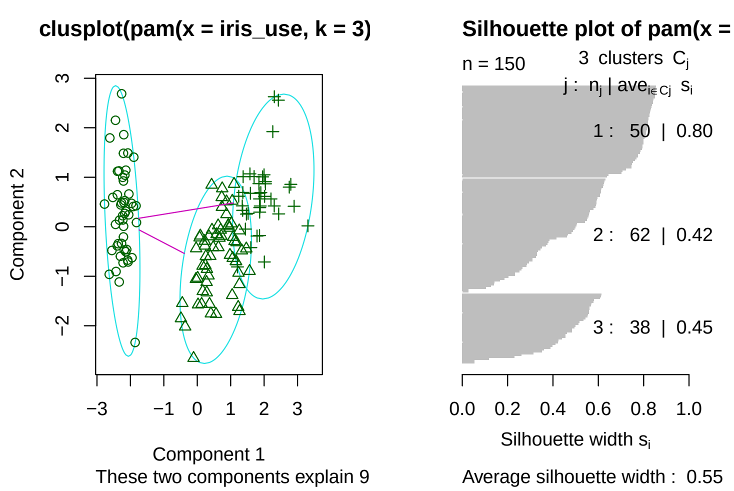

Figure 11.4: Analisis pengelompokan metode pam data iris.

11.10 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press.
2. Coqhlan, A. Tanpa Tahun. **Using R for Multivariate Analysis**. https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/src/multivariateanalysis.html#principal-component-analysis.
3. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung
4. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.
5. Rosidi, M. 2019. **Uji Hipotesis**. https://environmental-data-modeling.netlify.com/tutorial/11_uji_hipotesis/.
6. STHDA. Tanpa Tahun. **Comparing Means in R**. http://www.sthda.com/english/wiki/comparing-means-in-r.

Chapter 12

Pemodelan Data

Pada Chapter 12, kita akan membahas cara membentuk model statistik menggunakan `R`. Terdapat 2 buah jenis model yang akan dibahas pada *Chapter* ini, yaitu: regresi dan klasifikasi. Untuk informasi terkait cara untuk melakukan inferensi berdasarkan hasil yang diperoleh dan cara untuk melakukan prediksi menggunakan model yang terbentuk tidak akan dijelaskan dalam buku ini. Pembaca dapat membaca lebih lanjut pada referensi berikut:

- Introduction to Probability and Statistics Using R
- STHDA
- An Introduction to Statistical Learning

12.1 Regresi Linier

Regresi linier merupakan model sederhana yang paling sering dibahas dalam buku-buku statistika. Modelnya cukup sederhana dimana kita berusaha membentuk model dengan pendekatan garis linier dengan prinsip meminimalkan jumlah kuadrat residual pada data. Model yang tebentuk akan menghasilkan dua buah nilai yaitu nilai konstanta (titik potong sumbu y) dan nilai slope kurva. Model yang terbentuk secara umum haruslah memenuhi asumsi dasar model linier berikut:

1. **Asumsi liniearitas**: kurva relasi yang terbentuk antara variabel independen terhadap variabel dependen harus linier. Asumsi ini dapat dipelajari melalui plot residual terhadap nilai *fitted value*. Jika asumsi liniearitas terpenuhi, maka titik-titik residual yang di plotkan akan membentuk pola acak. Jika pada plot yang dihasilkan terbentuk pola tidak linear maka transformasi data pada variabel prediktor atau independen diperlukan.

2. **Error atau residu berdristribusi normal**: normalitas error di cek menggunakan qq-plot atau uji normalitas yang telah dibahas pada Chapter 11.4.
3. ***Outlier* dan *high influence point***: kedua pengamatan tersebut dideteksi melalui qq-plot, plot residual terhadap nilai *fitted value*, dan plot *residuals vs leverage*. Jika *outlier* terjadi akibat adanya error selama pengukuran maka *outlier* dapat dihilangkan.
4. **Error bersifat independen**: independensi residual dapat dideteksi melaui plot korelasi serial dengan mengeplotkan r_i vs r_{i-1}.
5. **Varians bersifat konstan**: Varians bersifat konstan dicek melalui plot **square root standardize residual vs fitted value**. Pada kasus dimana varians tidak bersifat konstan, kita dapat memberikan bobot pada model yang akan kita bentuk (*weighted least square*), dimana bobot yang diberikan proporsional dengan invers varians.
6. **multikolinearitas**: tidak ada variabel dependen yang saling berfkorelasi. Multikolinearitas dapat dideteksi melalui plot matriks korelasi. Pada model adanya kolinearitas ditunjukkan dari nilai *variance inflation factor* (VIF) yang tinggi. Secara umum nilai VIF terkecil sebesar 1 dan jika kolinearitas terjadi nilainya dapat lebih besar dari 5 atau 10. Untuk mengatasi kolinearitas pada model dapat dilakukan dengan dua cara, yaitu: mengeluarkan variabel dengan nilai VIF yang tinggi pada model atau menggabungkan dua variabel prediktor yang saling berkorelasi menjadi satu variabel baru.

Pembentukan model linier pada `R` dilakukan dengan menggunakan fungsi `lm()`. Format umum fungsi tersebut adalah sebagai berikut:

```
lm(formula, data, subset, weights)
```

Catatan:

- `formula` : formula model yang hendak dibentuk.
- `data`: data yang digunakan untuk membentuk model.
- `subset` : subset data yang akan digunakan dalam pembentukan model.
- `weight` : nilai pembobotan dalam pembentukan model.

12.1.1 Regrasi Linier Sederhana (*Simple Linear Regression*)

Pada Chapter 12.1.1 akan diberikan contoh pembentukan model linier sederhana menggunakan dataset `Boston` dari *library* `MASS` dengan jumlah observasi sebesar 506 observasi. Pada contoh kali ini kita akan mencoba membentuk model dengan variabel dependen berupa `medv` (median harga rumah) dan variabel independen berupa `lstat` (persen rumah tangga dengan status ekonomi menengah ke bawah). Berikut adalh sintaks untuk membentuk model tersebut:

```
library(MASS)
```

```
lm.fit <- lm(medv~lstat, data=Boston)
anova(lm.fit)
```

```
## Analysis of Variance Table
##
## Response: medv
##            Df Sum Sq Mean Sq F value Pr(>F)
## lstat       1  23244   23244     602 <2e-16 ***
## Residuals 504  19472      39
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.17  -3.99  -1.32   2.03  24.50
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  34.5538     0.5626    61.4   <2e-16 ***
## lstat        -0.9500     0.0387   -24.5   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.22 on 504 degrees of freedom
## Multiple R-squared:  0.544,  Adjusted R-squared:  0.543
## F-statistic:  602 on 1 and 504 DF,  p-value: <2e-16
```

Plot residual disajikan pada Gambar 12.1.

Berdasarkan hasil plot dapat dilihat bahwa seluruh asumsi model linier tidak terpenuhi. Selain melalui plot residual, uji asumsi model linier dapat juga dilakukan secara matematis. Berikut adalah sintaks yang digunakan:

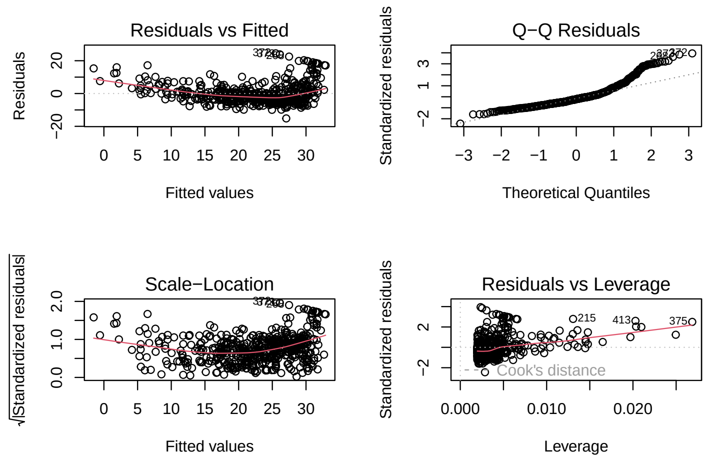

Figure 12.1: Analisis residual model linier medv vs lstat pada dataset Boston.

```
# error berdistribusi normal
# (data tidak berdistribusi normal)
shapiro.test(residuals(lm.fit))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(lm.fit)
## W = 0.88, p-value <2e-16
```

```
# varians bersifat konstan
# (varians tidak konstan)
library(lmtest)
bptest(lm.fit)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  lm.fit
## BP = 15, df = 1, p-value = 8e-05
```

```
# error bersifat independen
# (error tidak bersifat independen)
dwtest(lm.fit, alternative = "two.sided")
```

```
##
##  Durbin-Watson test
##
## data:  lm.fit
## DW = 0.89, p-value <2e-16
## alternative hypothesis: true autocorrelation is not 0
```

```
# deteksi outlier (stdres > 2)
sres <- rstandard(lm.fit)
sres[which(abs(sres)>2)] # nomor observasi outlier
```

```
##     99    142    162    163    164    167    181
##  2.038  2.038  2.759  2.787  3.000  3.058  2.003
##    187    196    204    205    215    225    226
##  3.172  2.947  2.833  2.934  2.789  2.287  3.200
##    229    234    257    258    262    263    268
##  2.560  2.822  2.001  3.274  2.488  3.201  3.628
##    281    283    284    369    370    371    372
##  2.326  2.308  2.976  2.991  3.063  2.946  3.946
##    373    375    413    506
##  3.847  2.498  2.601 -2.444
```

```
# influential observation
# observasi > percentil 50
# tidak ada observasi dengan jarak cook yang extrim
cooksD <- cooks.distance(lm.fit)
p50 <- qf(0.5, df1=2, df2=560-2)
any(cooksD>p50)
```

```
## [1] FALSE
```

12.1.2 Regresi Linier Berganda (*Multiple Linier Regression*)

Pada Chapter 12.1.2, kita akan membuat tiga buah model regresi linier. Model pertama akan menambahkan variabel `age` (usia bangunan) pada model sebelumnya, model kedua akan menggunakan seluruh ariabel yang ada, dan model ketiga akan melakukan pembaharuan dengan mengeluarkan variabel dengan VIF paling tinggi dari model kedua. Berikut adalah sintaks untuk membentuk ketiag model tersebut:

```
library(car)
# Model pertama
```

```r
lm.fit1 <- lm(medv ~ lstat+age, data=Boston)
anova(lm.fit1)
```

```
## Analysis of Variance Table
##
## Response: medv
##            Df Sum Sq Mean Sq F value Pr(>F)
## lstat       1  23244   23244  609.95 <2e-16 ***
## age         1    304     304    7.98 0.0049 **
## Residuals 503  19168      38
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(lm.fit1)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.98  -3.98  -1.28   1.97  23.16
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  33.2228     0.7308   45.46   <2e-16 ***
## lstat        -1.0321     0.0482  -21.42   <2e-16 ***
## age           0.0345     0.0122    2.83   0.0049 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.17 on 503 degrees of freedom
## Multiple R-squared:  0.551,  Adjusted R-squared:  0.549
## F-statistic:  309 on 2 and 503 DF,  p-value: <2e-16
```

```r
vif(lm.fit1)
```

```
## lstat   age
## 1.569 1.569
```

Berdasarkan hasil perhitungan diketahui nilai VIF dari model < 10, sehingga asumsi multikolinearitas terpenuhi. Untuk asumsi lainnya dapat dicek pada plot residual yang ditampilkan pada Gambar 12.2.

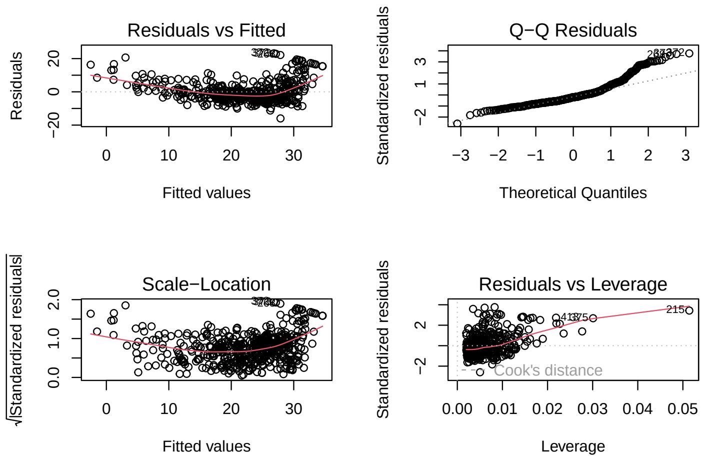

Figure 12.2: Analisis residual model linier 1 pada dataset Boston.

```
# Model 2
lm.fit2 <- lm(medv~., data=Boston)
anova(lm.fit2)
```

```
## Analysis of Variance Table
##
## Response: medv
##            Df Sum Sq Mean Sq F value  Pr(>F)
## crim        1   6441    6441  286.03 < 2e-16 ***
## zn          1   3554    3554  157.85 < 2e-16 ***
## indus       1   2551    2551  113.30 < 2e-16 ***
## chas        1   1530    1530   67.94 1.5e-15 ***
## nox         1     76      76    3.39 0.06635 .
## rm          1  10938   10938  485.75 < 2e-16 ***
## age         1     90      90    4.01 0.04581 *
## dis         1   1780    1780   79.03 < 2e-16 ***
## rad         1     34      34    1.52 0.21883
## tax         1    330     330   14.64 0.00015 ***
## ptratio     1   1309    1309   58.15 1.3e-13 ***
## black       1    593     593   26.35 4.1e-07 ***
## lstat       1   2411    2411  107.06 < 2e-16 ***
## Residuals 492  11079      23
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(lm.fit2)
```

```
##
## Call:
## lm(formula = medv ~ ., data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.594  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.65e+01   5.10e+00    7.14  3.3e-12 ***
## crim        -1.08e-01   3.29e-02   -3.29  0.00109 **
## zn           4.64e-02   1.37e-02    3.38  0.00078 ***
## indus        2.06e-02   6.15e-02    0.33  0.73829
## chas         2.69e+00   8.62e-01    3.12  0.00193 **
## nox         -1.78e+01   3.82e+00   -4.65  4.2e-06 ***
## rm           3.81e+00   4.18e-01    9.12  < 2e-16 ***
## age          6.92e-04   1.32e-02    0.05  0.95823
## dis         -1.48e+00   1.99e-01   -7.40  6.0e-13 ***
## rad          3.06e-01   6.63e-02    4.61  5.1e-06 ***
## tax         -1.23e-02   3.76e-03   -3.28  0.00111 **
## ptratio     -9.53e-01   1.31e-01   -7.28  1.3e-12 ***
## black        9.31e-03   2.69e-03    3.47  0.00057 ***
## lstat       -5.25e-01   5.07e-02  -10.35  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.75 on 492 degrees of freedom
## Multiple R-squared:  0.741,  Adjusted R-squared:  0.734
## F-statistic:  108 on 13 and 492 DF,  p-value: <2e-16
```

```r
vif(lm.fit2)
```

```
##    crim      zn   indus    chas     nox      rm
##   1.792   2.299   3.992   1.074   4.394   1.934
##     age     dis     rad     tax ptratio   black
##   3.101   3.956   7.484   9.009   1.799   1.349
##   lstat
##   2.941
```

Berdasarkan hasil perhitungan diperoleh nilai VIF untuk seluruh varaibel prediktor dalam model < 10, sehingga asumsi multikolinearitas terpenuhi.

Untuk asumsi lainnya dapat dicek pada plot residual yang ditampilkan pada Gambar 12.3.

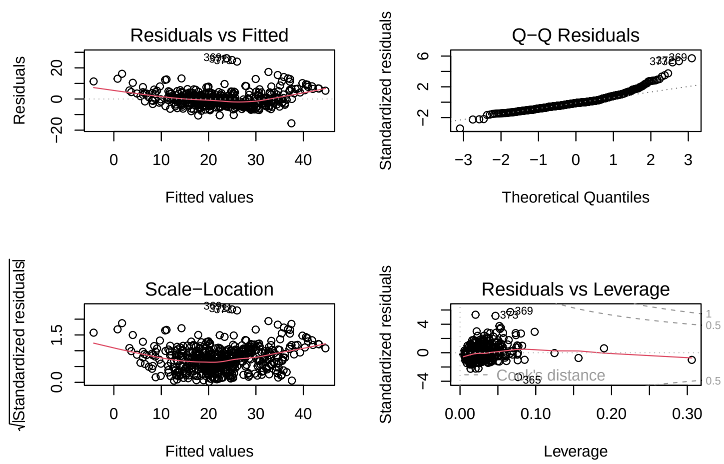

Figure 12.3: Analisis residual model linier 2 pada dataset Boston.

Pada model ketiga, kita akan mencoba untuk melakukan pembaharuan pada model kedua dengan melakukan drop variabel dengan vif yang paling tinggi. Pada hasil perhitungan sebelumnya, variabel `tax` (pajak) memiliki nilai VIF yang paling tinggi, sehingga pada model ketiga variabel tersebut tidak disertakan. Terdapat dua cara untuk melakukannya berikut adalah sintaks yang digunakan:

```
# Model 3 (cara 1)
lm.fit3 <- lm(medv~.-tax, data=Boston)

# Model 3 (cara 2)
lm.fit3 <- update(lm.fit2, ~.-tax)

anova(lm.fit3)
```

```
## Analysis of Variance Table
##
## Response: medv
##              Df Sum Sq Mean Sq F value  Pr(>F)
## crim          1   6441    6441  280.48 < 2e-16 ***
## zn            1   3554    3554  154.78 < 2e-16 ***
## indus         1   2551    2551  111.10 < 2e-16 ***
## chas          1   1530    1530   66.62 2.8e-15 ***
```

```
## nox         1     76      76    3.32   0.069 .
## rm          1  10938   10938  476.32 < 2e-16 ***
## age         1     90      90    3.93   0.048 *
## dis         1   1780    1780   77.49 < 2e-16 ***
## rad         1     34      34    1.49   0.223
## ptratio     1   1401    1401   61.02 3.4e-14 ***
## black       1    612     612   26.63 3.6e-07 ***
## lstat       1   2388    2388  103.99 < 2e-16 ***
## Residuals 493  11321      23
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(lm.fit3)
```

```
##
## Call:
## lm(formula = medv ~ crim + zn + indus + chas + nox + rm + age +
##     dis + rad + ptratio + black + lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.145  -2.914  -0.566   1.744  26.311
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.46e+01   5.12e+00    6.76  3.9e-11 ***
## crim        -1.07e-01   3.32e-02   -3.22  0.00138 **
## zn           3.64e-02   1.35e-02    2.69  0.00735 **
## indus       -6.78e-02   5.58e-02   -1.21  0.22532
## chas         3.03e+00   8.64e-01    3.51  0.00049 ***
## nox         -1.87e+01   3.85e+00   -4.86  1.6e-06 ***
## rm           3.91e+00   4.21e-01    9.29  < 2e-16 ***
## age         -6.05e-04   1.33e-02   -0.05  0.96380
## dis         -1.49e+00   2.01e-01   -7.39  6.3e-13 ***
## rad          1.35e-01   4.13e-02    3.26  0.00118 **
## ptratio     -9.85e-01   1.32e-01   -7.48  3.5e-13 ***
## black        9.55e-03   2.71e-03    3.52  0.00047 ***
## lstat       -5.22e-01   5.12e-02  -10.20  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.79 on 493 degrees of freedom
## Multiple R-squared:  0.735,  Adjusted R-squared:  0.729
```

```
## F-statistic:  114 on 12 and 493 DF,  p-value: <2e-16
```

```r
vif(lm.fit3)
```

```
##    crim      zn   indus    chas     nox      rm
##   1.792   2.184   3.226   1.058   4.369   1.923
##     age     dis     rad ptratio   black   lstat
##   3.098   3.954   2.837   1.789   1.348   2.941
```

Plot residual ditampilkan pada Gambar 12.4.

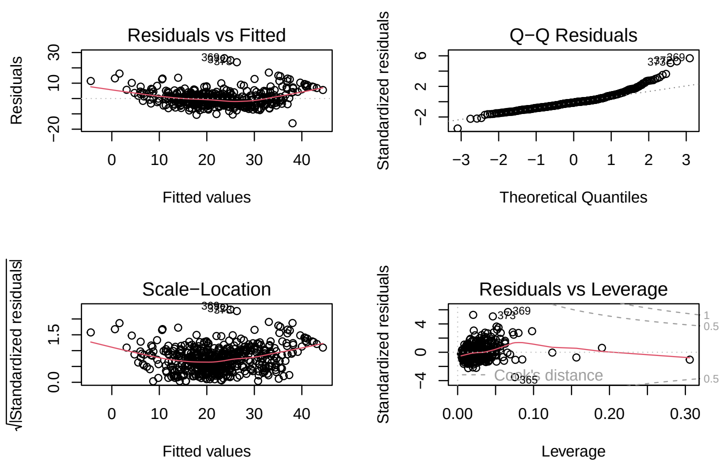

Figure 12.4: Analisis residual model linier 3 pada dataset Boston.

12.1.3 Model Linier dengan Interaksi Antar Variabel Prediktor

Interaksi antar variabel pada model linier dapat dengan mudah dimasukkan kedalam fungsi `lm()`. Terdapat dua buah cara untuk melakukannya. Cara pertama dengan menggunakan tanda `:` pada formula (contoh: $y1\ x1+x2+x1 : x2$). Tanda `:` menyatakan formula persamaan linier memasukkan interaksi antar variabel prediktor di dalamnya. Cara kedua adalah dengan menggunakan tanda `*`. Cara ini lebih sederhana, dimana fungsi `lm()` akan secara otomatis menerjemahkannya sebagai serangkaian variabel tunggal dan interaksinya. Berikut adalah contoh penerapannya menggunakan kedua cara tersebut:

```r
# cara 1
lm.inter <- lm(medv~lstat+age+lstat:age, data=Boston)
anova(lm.inter)
```

```
## Analysis of Variance Table
##
## Response: medv
##            Df Sum Sq Mean Sq F value Pr(>F)
## lstat       1  23244   23244  614.85 <2e-16 ***
## age         1    304     304    8.05 0.0047 **
## lstat:age   1    190     190    5.04 0.0252 *
## Residuals 502  18978      38
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(lm.inter)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age + lstat:age, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.81  -4.04  -1.33   2.08  27.55
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.088536   1.469835   24.55  < 2e-16 ***
## lstat       -1.392117   0.167456   -8.31  8.8e-16 ***
## age         -0.000721   0.019879   -0.04    0.971
## lstat:age    0.004156   0.001852    2.24    0.025 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.15 on 502 degrees of freedom
## Multiple R-squared:  0.556,  Adjusted R-squared:  0.553
## F-statistic:  209 on 3 and 502 DF,  p-value: <2e-16
```

```r
# Cara 2
lm.inter <- lm(medv~lstat*age, data=Boston)
anova(lm.inter)
```

```
## Analysis of Variance Table
##
## Response: medv
##            Df Sum Sq Mean Sq F value Pr(>F)
## lstat       1  23244   23244  614.85 <2e-16 ***
## age         1    304     304    8.05 0.0047 **
## lstat:age   1    190     190    5.04 0.0252 *
## Residuals 502  18978      38
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(lm.inter)
```

```
##
## Call:
## lm(formula = medv ~ lstat * age, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.81  -4.04  -1.33   2.08  27.55
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.088536   1.469835   24.55  < 2e-16 ***
## lstat       -1.392117   0.167456   -8.31  8.8e-16 ***
## age         -0.000721   0.019879   -0.04    0.971
## lstat:age    0.004156   0.001852    2.24    0.025 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.15 on 502 degrees of freedom
## Multiple R-squared:  0.556,  Adjusted R-squared:  0.553
## F-statistic:  209 on 3 and 502 DF,  p-value: <2e-16
```

Plot residual ditampilkan pada Gambar 12.5.

12.1.4 Transformasi Non-linier Pada Prediktor

Fungsi `lm()` juga dapat melibatkan transformasi non-linier prediktor pada argumen `formula`-nya. Transformasi non-linier dilakukan dengan menambahkan fungsi identitas `I()`. Sebagai contoh model berikut melibatkan transformasi kuadrat pada variabel `lstat`:

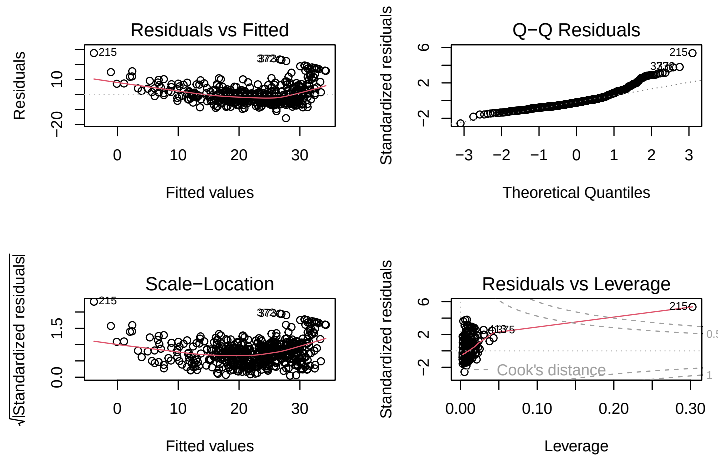

Figure 12.5: Analisis residual model dengan melibatkan interaksi antar variabel pada dataset Boston.

```
lm.trans <- lm(medv~lstat+I(lstat^2), data=Boston)
anova(lm.trans)
```

```
## Analysis of Variance Table
##
## Response: medv
##              Df Sum Sq Mean Sq F value Pr(>F)
## lstat         1  23244   23244     762 <2e-16 ***
## I(lstat^2)    1   4125    4125     135 <2e-16 ***
## Residuals   503  15347      31
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(lm.trans)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.28  -3.83  -0.53   2.31  25.41
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.86201    0.87208    49.1   <2e-16 ***
## lstat       -2.33282    0.12380   -18.8   <2e-16 ***
## I(lstat^2)   0.04355    0.00375    11.6   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.52 on 503 degrees of freedom
## Multiple R-squared:  0.641,  Adjusted R-squared:  0.639
## F-statistic:  449 on 2 and 503 DF,  p-value: <2e-16
```

Cara yang lebih sederhana untuk melibatkan tranformasi polinomial kedalam model linier adalah dengan menggunakan fungsi `poly()`. Berikut adalah contoh penerapannya:

```
lm.trans <- lm(medv~poly(lstat,2), data=Boston)
anova(lm.trans)
```

```
## Analysis of Variance Table
##
## Response: medv
##                 Df Sum Sq Mean Sq F value Pr(>F)
## poly(lstat, 2)   2  27369   13685     449 <2e-16 ***
## Residuals      503  15347      31
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(lm.trans)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 2), data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.28  -3.83  -0.53   2.31  25.41
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       22.533      0.246    91.8   <2e-16
## poly(lstat, 2)1 -152.460      5.524   -27.6   <2e-16
```

```
## poly(lstat, 2)2   64.227      5.524    11.6   <2e-16
##
## (Intercept)     ***
## poly(lstat, 2)1 ***
## poly(lstat, 2)2 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.52 on 503 degrees of freedom
## Multiple R-squared:  0.641,  Adjusted R-squared:  0.639
## F-statistic:  449 on 2 and 503 DF,  p-value: <2e-16
```

Plot residual ditampilkan pada Gambar 12.6.

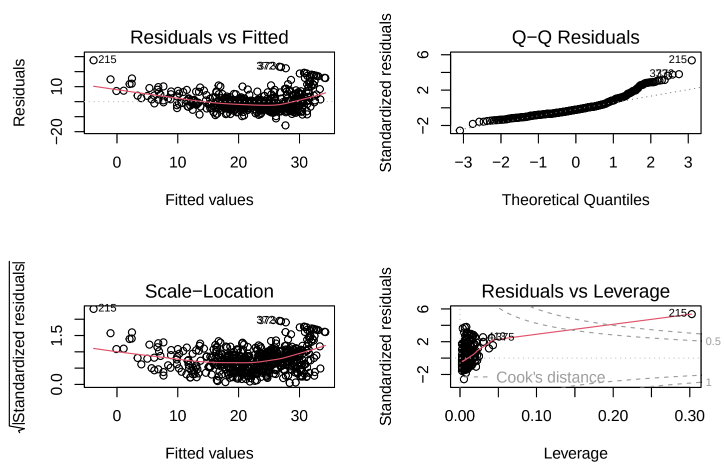

Figure 12.6: Analisis residual model dengan melibatkan transformasi non-linier variabel prediktor pada dataset Boston.

FUngsi trasnformasi lainnya juga dapat digunakan pada pembentukan model linier. Berikut adalah contoh penerapan transformasi logaritmik dan eksponensial pada model linier:

```r
summary(lm(medv~log(lstat), data=Boston))
```

```
##
## Call:
## lm(formula = medv ~ log(lstat), data = Boston)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.460  -3.501  -0.669   2.169  26.013
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   52.125      0.965    54.0   <2e-16 ***
## log(lstat)   -12.481      0.395   -31.6   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.33 on 504 degrees of freedom
## Multiple R-squared:  0.665,  Adjusted R-squared:  0.664
## F-statistic: 1e+03 on 1 and 504 DF,  p-value: <2e-16
```

```
summary(lm(medv~exp(lstat), data=Boston))
```

```
##
## Call:
## lm(formula = medv ~ exp(lstat), data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -17.57  -5.47  -1.37   2.43  27.43
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.26e+01   4.09e-01   55.19   <2e-16 ***
## exp(lstat)  -4.44e-16   2.79e-16   -1.59     0.11
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.18 on 504 degrees of freedom
## Multiple R-squared:  0.00501,    Adjusted R-squared:  0.00303
## F-statistic: 2.54 on 1 and 504 DF,  p-value: 0.112
```

12.1.5 Model Linier dengan Prediktor Kategorikal

Regresi linier dapat pula dilakukan dengan jenis variabel prediktor berupa variabel kategorikal. Untuk dapat melakukannya jenis data variabel kategorikal terlebih dahulu dirubah kedalam factor.

BELUM ADA contoh

12.1.6 Regresi linier dengan Pembobotan

Pada pembahasan sebelumnya kita telah menyaksikan bahwa sebagian model yang telah terbentuk tidak memenuhi asumsi varians yang konstan. Untuk mengatasi hal tersebut, kita dapat membentuk regresi dengan memberikan bobot sebesar invers variansnya. Untuk melakukannya diperlukan beberapa tahapan, antara lain:

1. Membentuk model linier dari variabel dataset: `fit <- lm(y~(variabel prediktor))`.
2. Menghitung error absolut (`abse <- abs(resid(fit))`) dan nilai *fitted value dari model (`yhat <- fitted(fit)`.
3. Membentuk kembali model menggunakan data residual absolut (`efit <- lm(abse~poly(yhat,2))`) dan menghitung *residual fitted value* (`shat <- fitted(efit)`).
4. Gunakan nilai bobot `w <- 1/shat^2` untuk membentuk model regresi dengan pembobotan (`fitw <-lm(y~(variabel prediktor), weights=w)`).

Kita akan membentuk kembali model menggunakan dataset `Boston` dengan menggunakan seluruh variabel, namun pada model kali ini kita akan memberikan bobot pada model yang terbentuk. Berikut adalah sintaks yang digunakan:

```r
# langkah 1
fit <- lm(medv~., data=Boston)

# langkah 2
abse <- abs(resid(fit))
yhat <- fitted(fit)

# langkah 3
efit <- lm(abse~poly(yhat,2))
shat <- fitted(efit)

# langkah 4
fitw <- lm(medv~., data = Boston, weights = 1/(shat^2))
anova(fitw)
```

```
## Analysis of Variance Table
##
## Response: medv
##            Df Sum Sq Mean Sq F value  Pr(>F)
## crim        1    367     367   188.5 < 2e-16 ***
## zn          1    148     148    76.1 < 2e-16 ***
```

```
## indus       1    125     125    63.9 9.3e-15 ***
## chas        1     41      41    21.2 5.2e-06 ***
## nox         1     38      38    19.7 1.1e-05 ***
## rm          1    363     363   186.6 < 2e-16 ***
## age         1     30      30    15.6 9.0e-05 ***
## dis         1    106     106    54.3 7.4e-13 ***
## rad         1      0       0     0.0    0.99
## tax         1     30      30    15.4 9.9e-05 ***
## ptratio     1     74      74    37.9 1.6e-09 ***
## black       1     64      64    33.1 1.6e-08 ***
## lstat       1    334     334   171.7 < 2e-16 ***
## Residuals 492    958       2
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(fitw)
```

```
##
## Call:
## lm(formula = medv ~ ., data = Boston, weights = 1/(shat^2))
##
## Weighted Residuals:
##    Min     1Q Median     3Q    Max
## -4.252 -0.795 -0.037  0.698  9.299
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  39.15845    4.47418    8.75  < 2e-16 ***
## crim         -0.13738    0.03254   -4.22  2.9e-05 ***
## zn            0.03975    0.01367    2.91  0.00380 **
## indus         0.03107    0.05211    0.60  0.55129
## chas          2.88679    0.81633    3.54  0.00044 ***
## nox         -15.84421    3.22177   -4.92  1.2e-06 ***
## rm            2.32247    0.41424    5.61  3.4e-08 ***
## age           0.00587    0.01147    0.51  0.60930
## dis          -1.13778    0.17774   -6.40  3.6e-10 ***
## rad           0.30255    0.05805    5.21  2.8e-07 ***
## tax          -0.01138    0.00328   -3.47  0.00056 ***
## ptratio      -0.72178    0.11598   -6.22  1.0e-09 ***
## black         0.00878    0.00234    3.75  0.00020 ***
## lstat        -0.62185    0.04746  -13.10  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 1.4 on 492 degrees of freedom
## Multiple R-squared:  0.642,  Adjusted R-squared:  0.633
## F-statistic:   68 on 13 and 492 DF,  p-value: <2e-16
```

Plot residual ditampilkan pada Gambar 12.7.

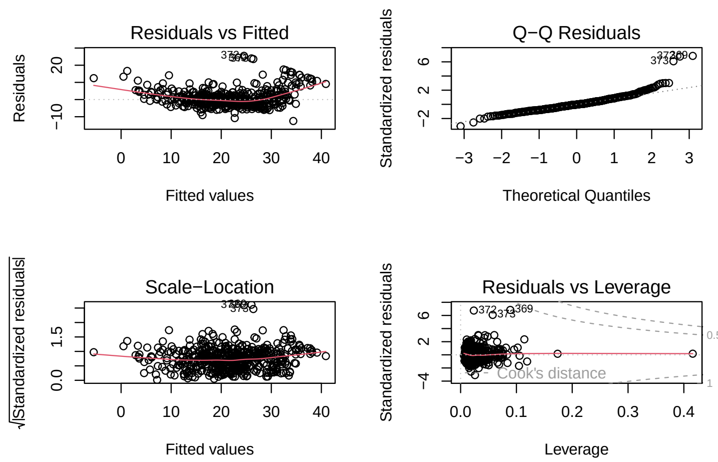

Figure 12.7: Analisis residual model regresi linier dengan pembobotan pada dataset Boston.

12.2 Regresi Logistik

Pada Chapter 12.2, kita telah membahas cara untuk membangun model dengan output berupa variabel dengan nilai numerik. Pada Chapter 12.2, kita akan belajar cara membentuk model regresi dengan 2 respons (0 dan 1). Pada regresi ini pembentukan model didasarkan oleh kurva logistik, dimana melalui kurva tersebut nilai yang dihasilkan akan memiliki rentang dari 0 sampai 1. Karena model yang dibuat bertujuan untuk memprediksi dua buah kemungkinan (0 atau 1), maka diperlukan suatu nilai ambang (y<0,5 = 0 dan y >= 0,5 = 1).

Fungsi `glm()` dapat digunakan untuk membentuk model regresi logistik. Format umum fungsi tersebut adalah sebagai berikut:

```
glm(formula, family = gaussian, data, weights, subset,
    )
```

Catatan:

- `formula` : formula model yang hendak dibentuk.
- `family` : distribusi yang digunakan. Untuk regresi logistik digunakan argumen `family=binomial`
- `data`: data yang digunakan untuk membentuk model.
- `subset` : subset data yang akan digunakan dalam pembentukan model.
- `weight` : nilai pembobotan dalam pembentukan model.

Belum ADA Contoh

12.3 Referensi

1. Akritas, M. 2016. **PROBABILITY & STATISTICS WITH R FOR ENGINEERS AND SCIENTISTS**. Pearson.
2. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press.
3. James, G., Witten, D., Hastie, T., Tibshirani, R. 2013. **An Introduction to Statistical Learning**. Springer.
4. Kerns, G.J., 2018. **Introduction to Probability and Statistics Using R**. Course notes for University of Auckland Paper STATS 330. http://ipsur.r-forge.r-project.org/book/download/IPSUR.pdf.
5. Lee, A., Ihaka, R., Triggs, C. 2012. **ADVANCED STATISTICAL MODELLING**.
6. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
7. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.
8. STHDA. <(http://www.sthda.com/english/>